NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ENERGY-EFFICIENT UNDERWATER SURVEILLANCE BY MEANS OF HYBRID AQUACOPTERS

by

Chase H. Dillard

December 2014

Thesis Co-Advisors: Vladimir Dobrokhodov
Kevin Jones

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 2014	Master's Thesis

4. TITLE AND SUBTITLE
ENERGY-EFFICIENT UNDERWATER SURVEILLANCE BY MEANS OF HYBRID AQUACOPTERS

5. FUNDING NUMBERS
RWG2Y

6. AUTHOR(S) Chase H. Dillard

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Consortium for Robotics and Unmanned Systems Education and Research
Monterey, CA 93943-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

This thesis develops algorithms in support of a prototype hybrid air-water quadcopter platform: the "AquaQuad." We consider the scenario in which AquaQuads with underwater acoustic sensing capabilities are tracking a submerged target from the surface of the ocean using sparse distributed measurements.

Multiple nonlinear estimation filters are evaluated for the tracking scenario, resulting in the selection of the unscented Kalman filter (UKF). Geometric positioning effects on estimators are explored through analysis of the horizontal dilution of precision metric. The UKF is then implemented in real-time on quadrotors using time-difference of arrival pseudo-measurements in an instrumented Vicon lab space.

The AquaQuads will primarily drift, but possess battery-limited flight capabilities. To increase on-station time, we seek to maximize use of the environment. In addition to solar energy, we take advantage of ocean currents that traditional autonomous platforms seek to reject. A novel sampling-based approach for path-planning is therefore created and lab-tested. The new algorithm, Dead-Reckoning Rapidly-Exploring Random Tree Star (DR-RRT*), combines the infinite-time optimality guarantees of RRT* with the unique AquaQuad mobility requirements. The DR-RRT* develops obstacle-free paths to a goal by linking brief flight and energy-efficient drift segments together, resulting in an energy savings of 27 percent over direct flight.

14. SUBJECT TERMS AquaQuad, path planning, rapidly-exploring random tree, unscented Kalman filter, nonlinear estimation, time-difference of arrival, dilution of precision

15. NUMBER OF PAGES
155

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ENERGY-EFFICIENT UNDERWATER SURVEILLANCE BY MEANS OF HYBRID AQUACOPTERS

Chase H. Dillard
Lieutenant, United States Navy
B.S., Penn State University, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2014

Author: Chase H. Dillard

Approved by: Vladimir Dobrokhodov
Thesis Co-Advisor

Kevin Jones
Thesis Co-Advisor

Garth V. Hobson
Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis develops algorithms in support of a prototype hybrid air-water quadcopter platform: the "AquaQuad." We consider the scenario in which AquaQuads with underwater acoustic sensing capabilities are tracking a submerged target from the surface of the ocean using sparse distributed measurements.

Multiple nonlinear estimation filters are evaluated for the tracking scenario, resulting in the selection of the unscented Kalman filter (UKF). Geometric positioning effects on estimators are explored through analysis of the horizontal dilution of precision metric. The UKF is then implemented in real-time on quadrotors using time-difference of arrival pseudo-measurements in an instrumented Vicon lab space.

The AquaQuads will primarily drift, but possess battery-limited flight capabilities. To increase on-station time, we seek to maximize use of the environment. In addition to solar energy, we take advantage of ocean currents that traditional autonomous platforms seek to reject. A novel sampling-based approach for path-planning is therefore created and lab-tested. The new algorithm, Dead-Reckoning Rapidly-Exploring Random Tree Star (DR-RRT*), combines the infinite-time optimality guarantees of RRT* with the unique AquaQuad mobility requirements. The DR-RRT* develops obstacle-free paths to a goal by linking brief flight and energy-efficient drift segments together, resulting in an energy savings of 27 percent over direct flight.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AUV	autonomous unmanned vehicle
BOT	bearing-only tracking
CAVR	Center for Autonomous Vehicle Research
DL	disk loading
DOP	dilution of precision
DR-RRT*	dead-reckoning rapidly-exploring random tree star
EKF	extended Kalman filter
FOM	figure of merit
GDOP	geometric dilution of precision
GRV	Gaussian random variable
HDOP	horizontal dilution of precision
KF	Kalman filter
LBL	long baseline
MLD	mixed layer depth
MPPT	maximum power point tracking
NREL	National Renewable Energy Laboratory
PI	proportional-integral
PID	proportional-integral-derivative
PRM	probabilistic roadmap
RRT	rapidly-exploring random tree
RRT*	rapidly-exploring random tree star
SWAFS	Shallow Water Analysis and Forecast System
TDOA	time-difference of arrival
UDP	user datagram protocol
UERE	user equivalent range error
UKF	unscented Kalman filter
USBL	ultra short baseline
UT	unscented transformation
VTOL	vertical takeoff and landing

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to first thank my cherished wife for her love and understanding during this process and also my two wonderful daughters who always enjoyed talking about and seeing the “flying robots.” Their passion for learning drives my own.

I’d also like to acknowledge and thank Drs. Doug Horner and Noel DuToit for an invaluable experience learning about and operating unmanned underwater vehicles in some truly amazing environments.

Thank you to my co-advisor Dr. Kevin Jones for his help with aerospace concepts previously unfamiliar to me, and thank you to fellow student Matteo Monari for saving me untold hours of work in the CAVR laboratory with his assistance.

Lastly, I am deeply indebted to my advisor Dr. Vladimir Dobrokhodov for his tireless guidance and wisdom. It has truly been an honor and a pleasure.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Undersea dominance is a core mission area for the United States Navy. Despite our high level of sophistication and experience in anti-submarine warfare, prosecution of increasingly quiet submarines continues to be an open problem. The use of force multipliers like unmanned and autonomous vehicles can aid the Navy greatly and are representative of our historical efforts to be on the leading edge of technology. These platforms have the potential to conduct a wide variety of missions, providing tremendous operational flexibility, power projection, and intelligence gathering capability.

An autonomous unmanned vehicle (AUV) can be more broadly described as a robot. One definition of a robot is a goal-oriented machine that can sense, plan and act [1]. Each of these three core capabilities of sensing, planning, and acting is dependent upon the other. Sensing the environment leads to understanding it, and this is critical for planning. Once the plan is made, the robot must be directed to act on it. Much research has been conducted by control system engineers to develop sophisticated feedback algorithms, which typically direct this action through rejection of disturbances in the environment. This can come at a tremendous cost in energy expenditure.

Independent of their usage, one of the issues with all AUVs is that constrained onboard energy supply limits their operational endurance. This fundamentally restricts their ability to conduct sustained missions and requires greater human involvement in their operation. We seek a new solution to this challenge, one that views environmental disturbances as energy and incorporates them into the planning process while maintaining focus upon the overall mission objectives.

For a sea-going platform, one of these primary disturbances is ocean current. Yet there is significant motive power within it that can be utilized, and examples of this exist in nature. The acorn worm, a primitive organism once thought to exist only in shallow water, has recently been observed intentionally changing its buoyancy in order to rise into an advantageous deep-sea ocean current for transportation between feeding sites [2].

Consider also the yeti crab, whose prevalence in the Southern Ocean is attributed to the rapid conveyance afforded to their short-lived larvae by the Antarctic Circumpolar Current [3]. It is these biologic underpinnings that provide insight into the solution to our problem.

B. THESIS OVERVIEW

This thesis develops algorithms with wide applicability in support of a prototype hybrid air-water quadcopter platform: the "AquaQuad." We consider the scenario in which a flock of AquaQuads with underwater acoustic sensing capabilities are tracking a submerged target from the surface of the ocean using sparse distributed measurements. The measurements are assumed to be either time-stamped or perfectly synchronous. The group behavior is cooperative, as each member can leverage the significant communication capability (update rate, rich content, and distance) available to surface-based platforms.

The measurements taken by the AquaQuads can be in the form of range, bearing or time-difference of arrival (TDOA), depending upon the sensor and signal processing equipment used. These are common nonlinear measurements whose analytic solution would contain significant error. To overcome this, multiple nonlinear estimation filters are objectively evaluated in simulation to determine their suitability for the tracking scenario. The dependency of the estimation process on the quadrotors' geometric positioning is also explored through analysis of the horizontal dilution of precision (HDOP); a metric that is directly related to a more general Cramer-Rao bound. An unscented Kalman filter (UKF) is then implemented in real-time on quadrotors estimating the position and velocity of an object using TDOA pseudo-measurements in the instrumented Vicon lab space of the Naval Postgraduate School's Center for Autonomous Vehicle Research (CAVR).

Due to our objective of minimizing energy at the path planning stage and the projected constrained battery capacity, the AquaQuads will primarily drift while conducting their surveillance mission, but have limited flight capabilities for periodic repositioning. To increase on-station time we seek to maximize the use of energy present

in the environment. In addition to photovoltaic cells that incorporate solar energy, we take advantage of ocean currents that traditional autonomous platforms simply act against. A novel sampling-based approach is created for path planning based upon the existing rapidly-exploring random tree star (RRT*) approach. The new algorithm, titled dead-reckoning rapidly-exploring random tree star (DR-RRT*), combines the infinite-time optimality guarantees of RRT* with the unique mobility capabilities of the AquaQuad.

The DR-RRT* develops obstacle-free paths to a goal by linking brief flight and energy-efficient drift segments together with the objective of reducing required flight time down to short hops. It also incorporates HDOP in the planning process to ensure path feasibility for the tracking scenario. Near-optimal paths can be found that are still computationally feasible for onboard microcontroller implementation. A Monte Carlo experiment is conducted to develop statistics on the energy saved by this method compared to direct flight. Paths created by the DR-RRT* are then followed by flying quadrotors programmed with simulated drifting behavior, and the results are analyzed.

THIS PAGE INTENTIONALLY LEFT BLANK

II. THE AQUATIC QUADROTOR CONCEPT

A. AQUAQUAD OVERVIEW

The AquaQuad was born from the vision of integrating a number of existing technologies for propulsion, energy harvesting, signal processing, and communication into a single platform for tracking underwater targets. In doing so, we can create a novel, significantly more capable system, which may revolutionize existing missions and reveal new applications.

From a hardware standpoint, it is desirable to have a low-cost vehicle that is highly controllable, agile, and simple to maintain. Quadrotors possess all of these qualities. Rapid advances in wireless mesh communication and the minimal production cost of multicopters facilitate the use of multiple quadrotors, which opens the door to cooperative and swarm behaviors over a broad spectrum of missions. They have relatively high efficiency for vertical takeoff and landing (VTOL) platforms and can precisely control their attitude with motor speed changes. The brushless motors in each rotor have essentially no wear parts and are tolerant of wet environments. This makes them highly reliable with only four moving parts (quad-rotor configuration) and a rapidly growing commercial market for cheap parts and supplies.

The projected configuration of each AquaQuad includes photovoltaic cells affixed to the surface for absorbing electrical energy during daylight hours. They will also have an acoustic sensor that is connected to the underside of the platform. Figure 1 displays a rendering of this concept.

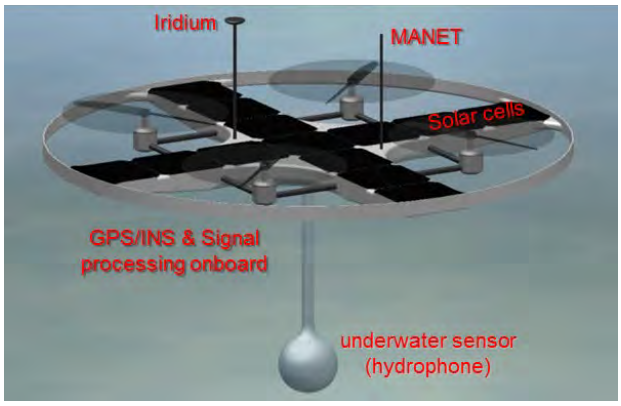

Figure 1. AquaQuad concept showing solar cells and hydrophone

Adaptation of a quadrotor to the proposed aquatic scenario requires some modifications. Parts of the vehicle must be encased in a water-tight enclosure to protect sensitive avionics components, and the NPS team is working to create an in-house version for future testing. Figure 2 shows an example of such an enclosure. The ocean environment is very dynamic, so the design of the AquaQuad must incorporate positive buoyancy and unconditional stability. The pendulum-like nature of the hydrophone attachment can assist with this self-righting behavior, in addition to the outrigger-style stability afforded by the rotor arms. The sensor itself is anticipated to be suspended from a flexible, retractable or fixed cable, the length of which could conceivably allow for acoustic searches inside of and below oceanic layers.

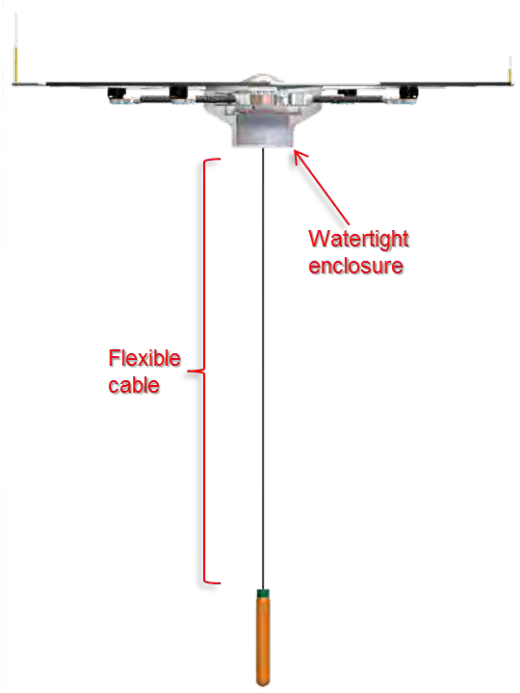

Figure 2. AquaQuad concept showing watertight enclosure and flexible cable

Once several of these AquaQuads are designed, they can be employed as a distributed sensor network. Figure 3 displays a drawing of this capability, the underlying concept of which is reflected in the rest of this work. The use of several dispersed sensors carries with it great operational flexibility, since they can be arranged and configured to maximize the group capability. Track of a target can be maintained using a single sensor, but the position error is greatly reduced by increasing the number of platforms sharing their individual measurements. This improves the accuracy of the final position estimate that can be used for weapon placement or for handoff to a more traditional ASW asset like a manned submarine, highlighting their implicit “force multiplier” capability.

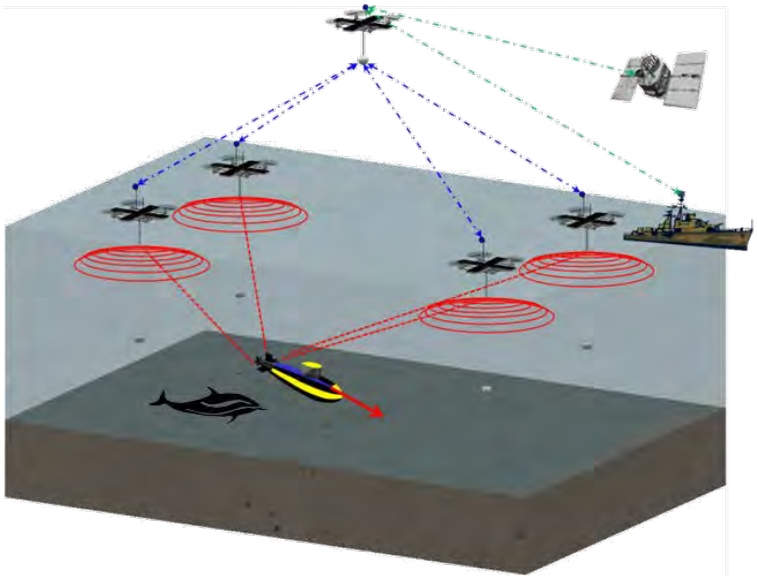

Figure 3. AquaQuad distributed sensor network concept

B. SUBMERGED TARGET TRACKING WITH SURFACE-BASED PLATFORMS

The use of surface-based (vice underwater) platforms for tracking submerged contacts has some major benefits but also carries with it some limitations. These limitations are principally related to the sensor's ability to identify acoustic signals in the ocean environment. Therefore some assumptions need to be made in their use.

1. Impact of Ocean Acoustics and Relevant Assumptions

In their current conceptual configuration, the AquaQuads have a potentially restricted ability to track deep-water acoustic signals. This is dependent upon the environmental conditions that exist in the operating area, and a brief discussion of these effects follow. In a generic sense, sound emitted from an underwater source consists of one or more pressure disturbances. These disturbances are mathematically modeled as waves propagating outwards from the source. Ray theory [4] provides a means of visualizing these waves and predicting their paths of travel by connecting lines normal to the direction of propagation between the wave fronts.

The paths that the rays follow are largely affected by the gradient of sound speed in the medium it is traveling through. Sound speed in the ocean is typically approximated

by a function of temperature, salinity and depth. Although there are many equations to relate these variables, one version is given in Equation (II.1) adapted from [4]

$$c = 1449.2 + 4.6T - 0.055T^2 + 0.00029T^3 + (1.34 - 0.010T)(S - 35) + 0.016z \quad \text{(II.1)}$$

where T = temperature (^{o}C), S = salinity (ppt), and z = depth (m). Through a simple analysis of Equation (II.1) it can be seen that sound speed increases whenever temperature, depth, or salinity increase. When combining the effects of these three variables, one can create a profile like that shown in Figure 4 from [4].

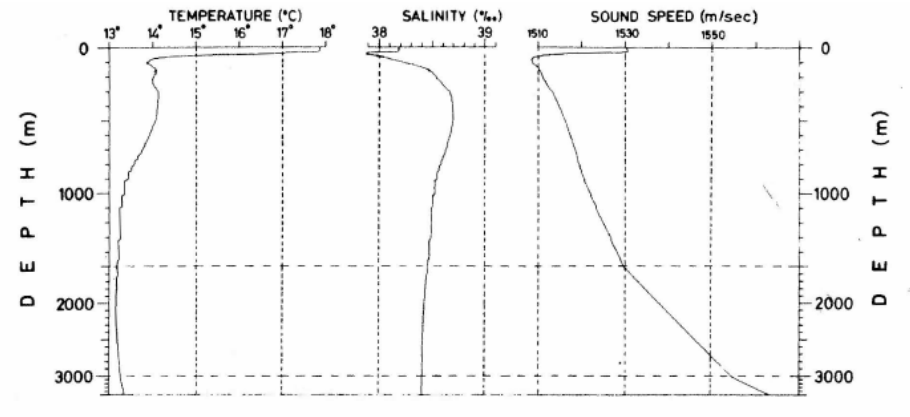

Figure 4. Environmental variations and the resultant sound speed profile, from [4]

The sound speed profile on the far right of Figure 4 exhibits a very typical pattern. As depth increases down the y-axis, there is a sharp decrease in temperature corresponding to the drop-off in sunlight penetration. The net effect of this temperature drop is a minimum in sound speed. Proceeding deeper, the pressure effects from depth begin to dominate, and the sound speed increases to a maximum.

With a sound speed profile known or assumed, it is possible to predict the paths the rays will take utilizing Snell's law, shown in Equation (II.2). This law states that the ratio between the ray's angle from the vertical axis and the sound speed at its point of departure are fixed.

$$\frac{\sin(\theta_1)}{c_1} = \frac{\sin(\theta_2)}{c_2} \ldots = \frac{\sin(\theta_n)}{c_n} \tag{II.2}$$

Therefore, if we know the sound speed at two different points and we make an assumption on the initial angle of departure, we can easily calculate the final angle the ray will make. When conducting this process repeatedly in a piecewise manner, the path of the ray can be approximated. Figure 5 from [4] shows the resulting plot of a ray tracing routine and in particular highlights the tendency for sound rays to bend towards areas of lower sound speed. This behavior focuses much of the sound energy into an axis corresponding to a sound speed minimum in the profile.

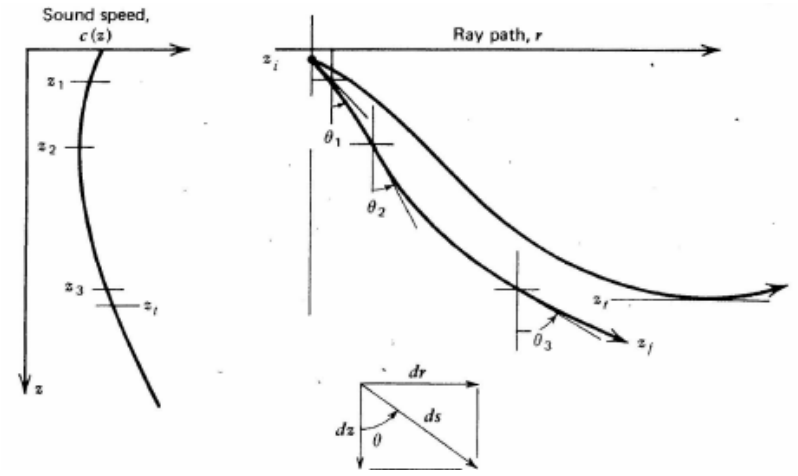

Figure 5. Basic ray tracing in a continuously stratified medium, from [4]

If the object we are attempting to track is operating at or near the deep-water sound speed minimum, it is likely that the majority of its sound energy will not be received by our sensors near the surface. Conceptually, this effect can be mitigated by utilizing AquaQuads equipped with variable depth sensor payloads; however we focus our attention instead on targets within the surface layer or "mixed layer" shown at the top of Figure 6 from [4]. The surface layer has a nearly uniform temperature distribution as a result of the mixing effects from wind forces and is a common phenomenon found throughout the world's oceans. Sound speed increases gradually with depth in the surface layer until it reaches the seasonal thermocline, often resulting in a local sound speed

minimum at the surface. Much of the sound energy produced within this layer will therefore propagate through it as the rays bend upward towards the surface, are reflected off of it, and the cycle repeats.

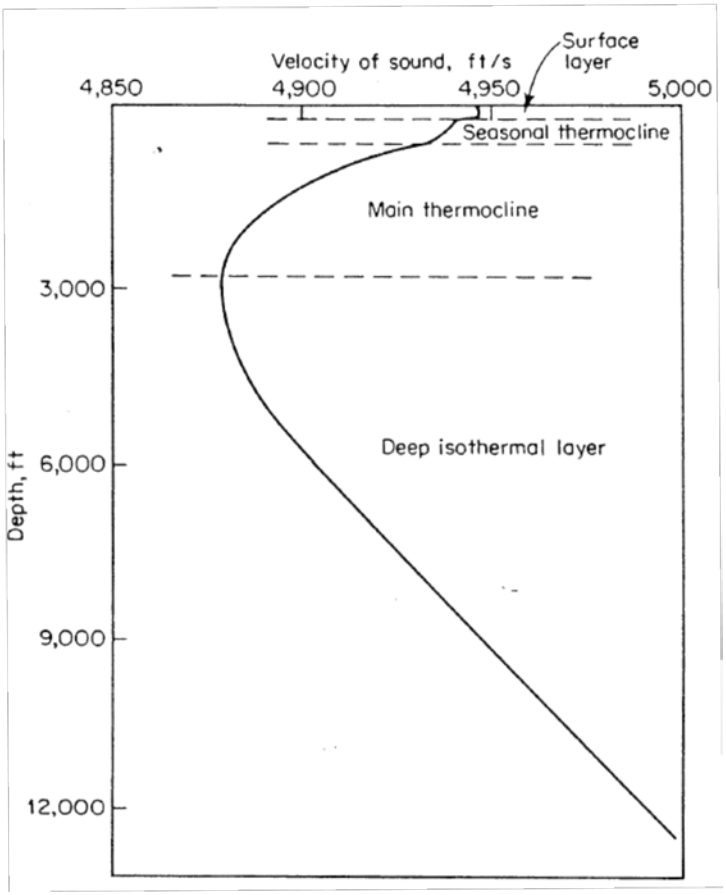

Figure 6. Sound speed profile displaying a surface layer, from [4]

The position of the surface layer, also known as the "mixed layer depth" (MLD), in a water column varies widely, but for comparison purposes Figure 7 shows a plot of the MLDs in the South China Sea from [5]. This figure shows depths on the order of 40 meters (131 feet). Sound energy generated within the mixed layer will propagate throughout it, and so we make the assumption that our target is relatively shallow.

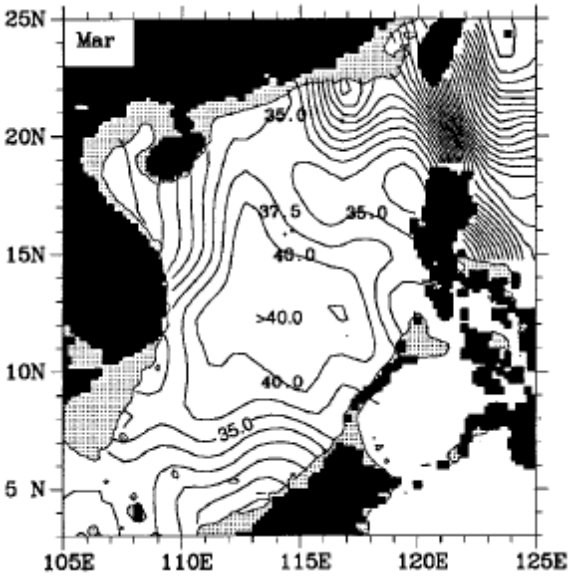

Figure 7. Estimated mixed layer depths (in meters) in the South China Sea, from [5]

In addition to this assumption's acoustic foundation, it also has an operational foundation. The majority of nations with undersea-capable navies utilize diesel-electric submarines. These submarines are often operated in the littoral regions, are required to recharge their batteries regularly with their diesel engines for extended periods of time ("snorkel") and must come shallow to do so. Snorkeling in particular is a noisy evolution that puts a large amount of sound energy into the mixed layer where the AquaQuad sensors will be present.

2. The AquaQuad Advantage

Despite their potential deep-water limitations, AquaQuads possess significant advantages when compared to their subsurface counterparts. At their most basic level, collaborative behaviors require communication, and underwater communications are inherently difficult and limited in range. The AquaQuad's position above the waterline facilitates air-based communication, allowing them to not be similarly encumbered. This is critical for collective maintenance of the target position estimate and dynamic repositioning of the individual platforms in order to achieve group objectives.

Knowledge of the location of each sensor in the flock is also vital to the accuracy of the Kalman filter target position estimate. Lacking an active acoustic long baseline (LBL) or ultra-short baseline (USBL) location system, undesirable for a covert application, the uncertainty of the position of an underwater vehicle grows with time, requiring it to break contact for a GPS fix. Conversely, each of the AquaQuad platforms is receiving continuous GPS signals. While drifting, the GPS can also provide an approximation of the regional ocean current. When shared between multiple sensors, this provides an update to the ocean current map that the flock will be using to minimize energy expenditure, improving the overall solution.

The use of quadrotors also allows for rapid repositioning at speeds higher than those typically seen in underwater platforms. This helps to ensure greater continuous coverage of the target being tracked and facilitates maintaining closer proximity to the moving target. Flight, however, carries a significant cost in energy consumption. This is a traditional constraint in autonomous vehicles and one we seek to overcome with the use of our hybrid platform.

C. AQUAQUAD ENERGY REQUIREMENTS

The use of the term "hybrid" when referring to the AquaQuad describes its ability to float on the surface of the ocean or, leveraging its VTOL capability, fly for repositioning. Our challenge then becomes one of maximizing on-station time by managing the finite battery capacity, and we approach this by first making some assumptions regarding the onboard energy budget.

Our first assumption is that the only source of energy replenishment will be via solar radiance. The prototype utilizes an array of 20 SunPower research-grade E60 monocrystalline silicon cells with an advertised efficiency of just over 24 percent. The array area of an envisioned prototype platform is roughly $0.3m^2$, yielding an ideal power output of about 73W. More important however is the amount of useable solar radiation in a 24 hour period. According to the National Renewable Energy Laboratory (NREL) PVWatts calculator, for the Lake Nacimiento, California (CA) area (a potential test site), the daily solar radiation peaks at $8.45kWh/m^2$ in June and falls to $2.56kWh/m^2$ in

December, for a horizontal array. With the given array size of $0.3m^2$, this corresponds to 608Wh in June and 184Wh in December. It should be noted that the actual solar radiation available will be time-of-day dependent and would likely follow a near-Gaussian curve like that shown in Figure 8, where integrating the area under the curve provides the energy available in a day. For the purposes of this thesis, we take the average of the two solar radiation values to arrive at approximately 400Wh of solar energy available in a 24-hour period. In order to prevent exhaustion of the onboard battery, 400Wh then becomes the amount of energy we have available for communications, sensors and flight.

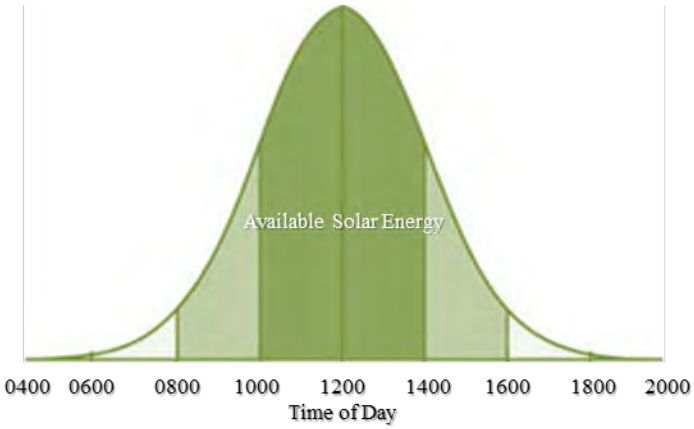

Figure 8. Gaussian-style distribution of available solar energy as a function of the time of day

Were the AquaQuads directed to fly continuously, solar energy could not alone sustain them. In a VTOL platform, the power required for flight is high, whereas the surface area available for photovoltaic cells is small. While hovering, the typical definition for propulsive efficiency fails so instead a figure of merit (FOM) can be used. In this case, we use grams of thrust per Watt (g/W) of electrical power as the FOM, which is plotted against disk loading in Figure 9. Disk loading (DL) is the weight of the vehicle divided by the area covered by the rotors, and from actuator disk theory it has an approximately inverse relationship with FOM [6]. The larger the FOM is, the greater the weight that can be lifted for a given Watt of power.

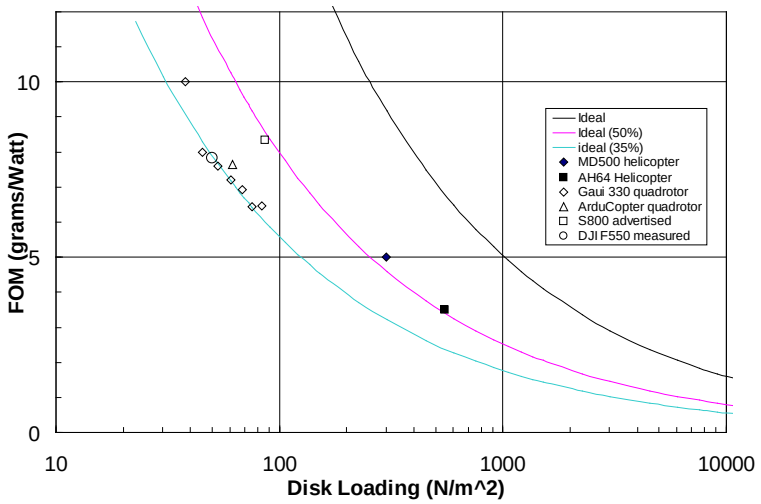

Figure 9. Experimentally obtained figure of merit as a function of disk loading for a variety of multi-rotor and helicopter platforms, compared with actuator disk theory.

The black line representing actuator disk theory in Figure 9 only considers waste momentum, ignoring all other losses. Due to mechanical, aerodynamic, and other losses in a real-world platform, this makes this theoretical bound a physical limit that can never truly be achieved. Consider the S800 hex-rotor aircraft [7] with a flying mass of 6kg. From Figure 9, the S800 requires 720W to operate or roughly 17,280Wh for 24 hours of flight. Even assuming the S800 received the maximum June solar radiation of 8.45kWh/m^2, the required array area would be approximately 8.5 square meters, therefore extended quadrotor flight is clearly not a possibility utilizing solar energy alone.

For a notional design of the AquaQuad platform, we estimate a weight of 2kg with a 10g/W FOM, corresponding to 200W of power required for flight. In addition, several other electrical loads exist that need to be accounted for. The first is a 5W base load estimate to keep critical systems in an active status. We also anticipate a 10W draw for intermittent sensing and communications. When considered over a 24-hour period, this results in 120Wh of base energy consumption and approximately 60Wh from the sensing load (since this equipment is not required at all times). Referring back to our original assumption that battery capacity will reflect available solar energy, starting out

with 400Wh yields 220Wh available for flight, corresponding to approximately one hour of flight time per day. This energy budged is tabulated in Table 1.

	Power	Time	Energy
Base	5W	24hrs	120Wh
Sensing & Comms	10W	6hrs	60Wh
Flight	200W	1.1hrs	220Wh
Solar	-	24hrs	400Wh

Table 1. Energy budget for the AquaQuad concept. Energy consumers are shown in red. Solar energy shown in black is the mean value of a 24-hour period.

D. UTILIZATION OF ENVIRONMENTAL ENERGY

With the limit on flight time so imposed, the AquaQuad requires additional means of repositioning itself to meet group objectives. To this end, we utilize the disturbances inherent to the operating environment. We do this by incorporating our understanding of prevailing ocean currents into the planning process. This allows us to intelligently plan periods where flight is required, while otherwise maximizing the amount of time being spent in a low-energy drift mode.

While drifting, the AquaQuads' motion will be almost entirely driven by the ocean surface currents that exist in their location. Due to their low profile, the wind effects are expected to be minimal. Each AquaQuad will be provided with an initial ocean current map based upon modeling software. As an example of such an application, consider the U.S. Navy's Shallow Water Analysis and Forecast System (SWAFS). It is a near real-time three-dimensional analysis and prediction tool that incorporates in situ measurements within an ocean forecast model commonly referred to as the Princeton model [8]. The unclassified resolution of the SWAFS model varies by region but can go as low as 1 nautical mile (nm). Figure 10 displays a low-resolution version of SWAFS' output with the type of current vector arrow overlay we are primarily concerned with.

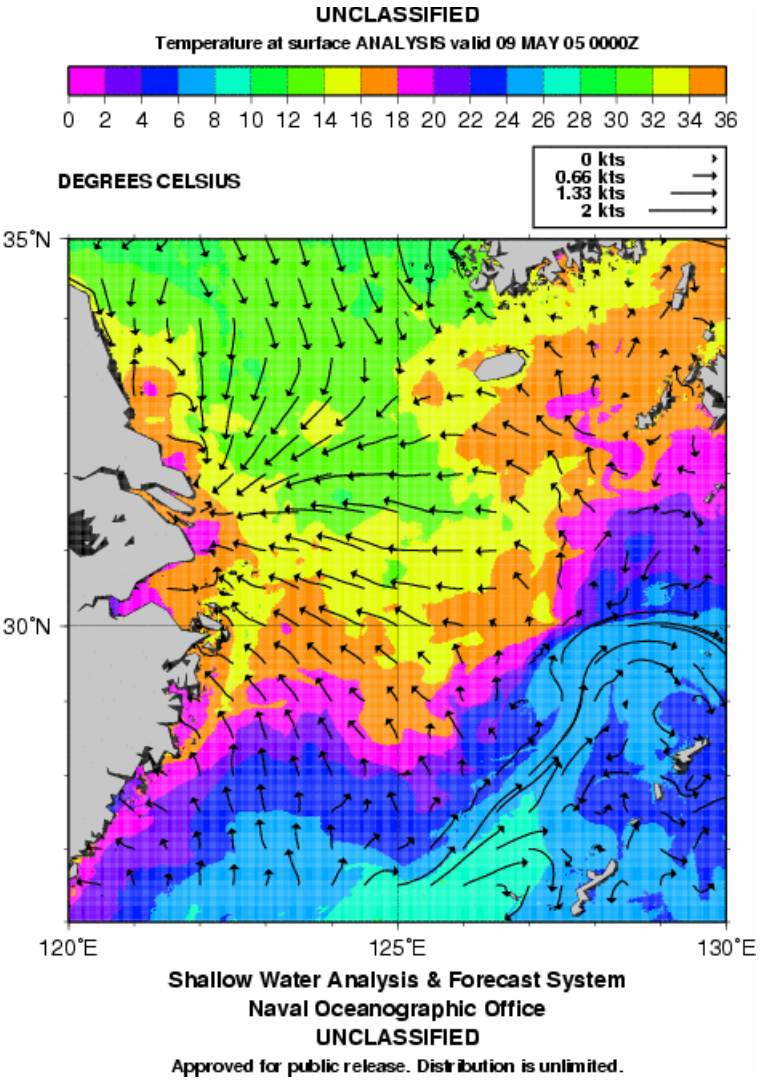

Figure 10. Shallow Water Analysis and Forecast System product, displaying regional ocean current vectors overlaying a sea surface temperature heat map [9].

By discretizing the data from these models into specific geographic cells, we can predict an AquaQuad's motion in that space. This prediction is understandably subject to the uncertainty of the map it is based upon; however the AquaQuad has the ability to communicate the true ocean current at its location for update by a host. This host can exist either at a remote ground high performance computing facility or as a local drifting AquaQuad equipped with long haul communication capability. With a preliminary understanding of the drift behavior expected from an AquaQuad placed in any starting location, we can use this information to search for the best combination of drift and flight

sequences that reaches a goal position. A concept of the desired behavior is shown in Figure 11, which displays a series of short flight "hops" that terminate in a drift to a final desired location.

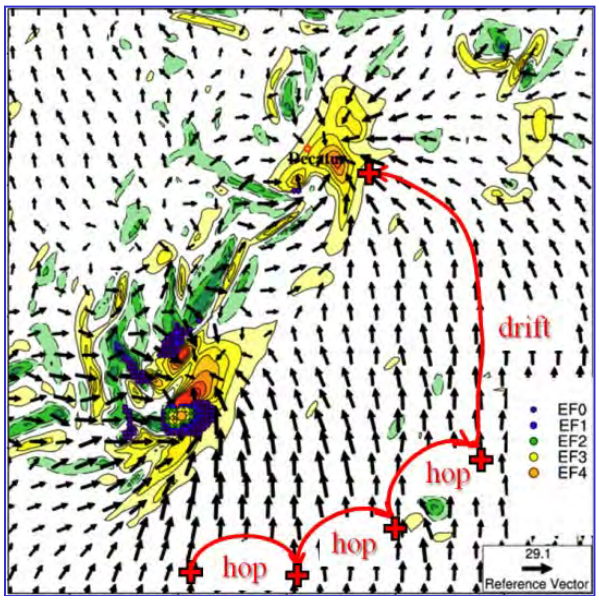

Figure 11. Path planning concept that utilizes predicted ocean current fields to minimize flight time in reaching a desired final position

The drifting steps in Figure 11 not only reduce energy expenditure from restricting flight time, they also incorporate energy gain from the solar cells during daylight hours. This planning process can be conducted near-optimally and will be described in Chapter IV.

E. POTENTIAL MEASUREMENT TYPES

Within our target position estimation filter, we will use one of three typical acoustic measurements from a target: bearing, range, or time-difference of arrival (TDOA). Each of these measurement types requires a specific sensing unit and processor to turn the raw pressure disturbances into a quantity that is relatable to the target position. Passive acoustic sensors are of particular importance to our application. Since a drifting AquaQuad with a passive hydrophone emits no sound energy into the environment, the chance of counter-detection is low. The battery power required to operate a passive

system is also significantly less than that required for active systems [10], leading to greater possible time on station. As the specific source of the measurement is not the primary focus of this thesis, we instead detail the fundamental equations that relate the measurement to our state and briefly describe some current methods of obtaining these measurements. These equations will be utilized within a generalized Kalman filter architecture as a means of predicting the next incoming measurement for comparison and are all a function of the Cartesian target position estimate $\hat{x}$ and $\hat{y}$.

1. Bearing Measurements

The bearing sensed by the AquaQuad is relative to the position of the platform itself (i.e., the quadrotor is at the center of a local coordinate system and the target position is defined relative to it). The bearing-only nonlinear measurement equation is simply

$$\theta_i = \tan^{-1}\left(\frac{\hat{y} - y_i}{\hat{x} - x_i}\right) \tag{II.3}$$

where y_i and x_i denote the position of the i^{th} AquaQuad under consideration. For implementation in MATLAB, the use of the *atan2* function is recommended, as it is the four-quadrant inverse tangent function that wraps the resultant bearing in radians from $[-\pi, \pi]$. Figure 12 illustrates the geometric configuration of these sensors in a two-dimensional XY plane using the notation from Equation (II.3).

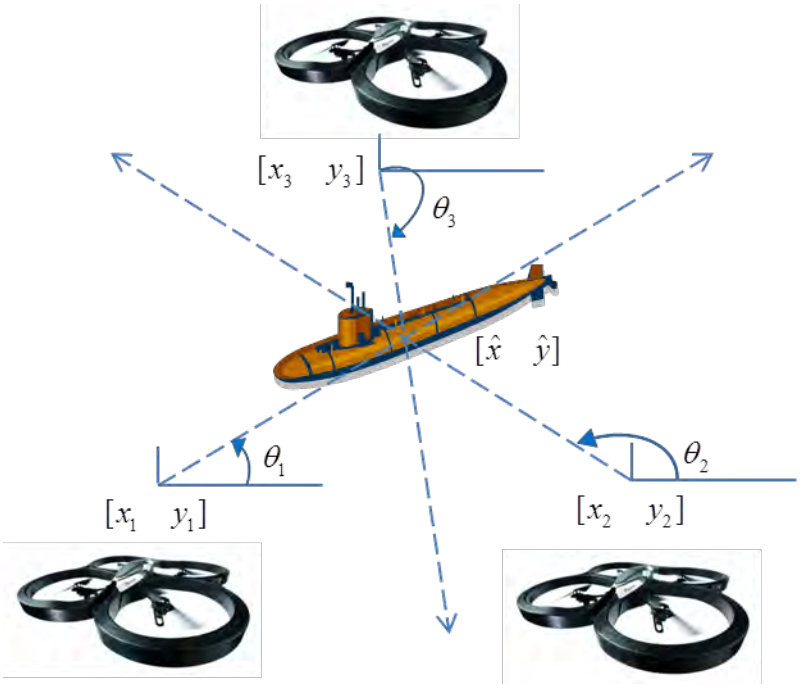

Figure 12. Bearing-only sensor two-dimensional geometric configuration

The measurement of the bearing of a signal is a basic but important task in underwater acoustics. In order to measure it, the properties of the impinging wave front must be sampled by a multi-sensor array, which takes advantage of the array's directionality [11]. Suspension of an array beneath the AquaQuad platform is possible but challenging due to payload restrictions on weight. Nevertheless, the bearings-only tracking (BOT) scenario is ubiquitous in target estimation problems and will help to illustrate important concepts in later chapters.

2. Range Measurements

The nonlinear measurement equation for range is the Cartesian coordinate distance equation, again defined with respect to the position of the quadrotor being analyzed.

$$r_i = \sqrt{(\hat{x} - x_i)^2 + (\hat{y} - y_i)^2} \tag{II.4}$$

Some interesting work has been conducted by [12] and [13] in determining the range of an underwater source, by using so-called waveguide invariant theory. When

looking at a spectrum of broadband energy (intensity) from a contact, one can see interference patterns due to the differences in sound propagation.

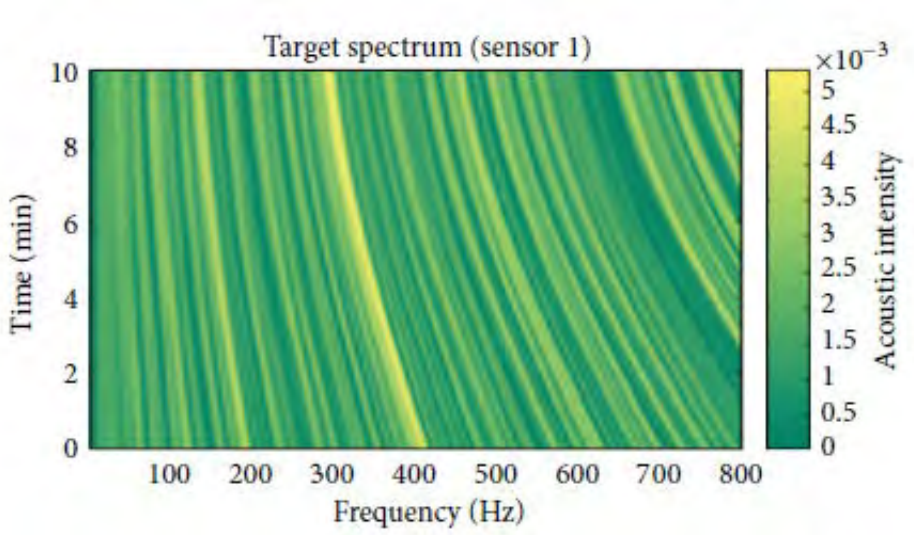

Figure 13. Interference patterns visualized as striations in a spectrogram plot of acoustic intensity, from [13]

In both papers, the slope of the interference pattern is described as a function of frequency and range, as

$$\frac{\partial f}{\partial r} = \beta \frac{f}{r} \tag{II.5}$$

The symbol β is the waveguide invariant, where the term "invariant" comes from the fact that, although the equation is unique to each mode pair, the numerical value of β will remain approximately the same. In [12], it is given the default value of 1. Simply solving the above equation for range *r* then provides the desired formula.

The slope of the interference pattern itself is dependent upon the frequency and range "window" that is being looked in. Thus, [12] takes the simplistic approach of measuring multiple windows and averaging the resultant range estimation. The authors of [12] also go into detail regarding the 2-D discrete Fourier transform process that is used to estimate the slope of the interference pattern automatically through image processing. Range estimates using this method are described as accurate within 25 percent in

simulation for source ranges between 500–2,200 meters and frequencies between 350 and 700 Hz [12].

The authors of [13] take a different approach to range estimation, explicitly taking into account multiple sensors. One received signal from a sensor is selected as a reference signal. The other sensors then have their received signal scaled to match the reference. The amount of scaling that is required to overlay the signals is said to be equal to a fixed ratio of range between the sensors and the source. This ratio will be constant even as the physical ranges themselves are changing. The position of the source is determined through the use of time delay estimation methods and the geometry seen in the circle of Apollonius, which follows.

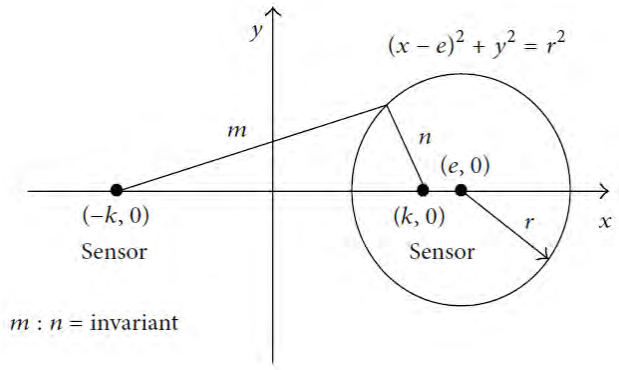

Figure 14. Circle of Apollonius used in determining source position, from [13]

Range estimates using this method are described as accurate within seven percent in simulation for source ranges between 400–2,000 meters and frequencies between 50 and 750 Hz [13].

3. Time Difference of Arrival

The time difference of arrival (TDOA) measurement is constructed from the difference between the times when two sensors, separated by some distance, receive the same signal. This value is derived from the basic relation that time is equal to distance divided by speed, the equation for which is

$$\tau_i = \frac{1}{c}\left(\sqrt{(\hat{x}-x_i)^2+(\hat{y}-y_i)^2}\right) \tag{II.6}$$

where *c* represents the speed of sound in water from Equation (II.1). When Equation (II.6) is extended to a time difference between two sensors denoted *i* and *j*, the TDOA measurement of Equation (II.7) from [14] is the result.

$$\tau_{ij} = \tau_i - \tau_j = \frac{1}{c}\left(\sqrt{(\hat{x}-x_i)^2+(\hat{y}-y_i)^2} - \sqrt{(\hat{x}-x_j)^2+(\hat{y}-y_j)^2}\right) \tag{II.7}$$

Equation (II.7) represents the half-hyperbola whose foci are the sensors themselves [14]. When multiple measurements are obtained, these hyperbolas cross and correspond to the position of the target.

Determining the time delay requires cross-correlation of the received signals. As an example of this process, consider Figure 16 and Figure 17. The data in these plots was collected in an August 2014 NPS experiment where two Acousonde™ acoustic sensors [15] (see its technical specification in the Appendix A), a lightweight (approximately 262g) and portable data logging device, received a sweep signal in the Monterey Bay.

Figure 15. Acousonde™ acoustic sensor utilized in time difference of arrival experiment

Once the sweep time was identified from the frequency display on the bottom of Figure 16, the signals were cross-correlated in Figure 17. The resultant lag in the cross-

correlation plot represents the TDOA measurement, with the sign convention representative of which sensor received the signal first.

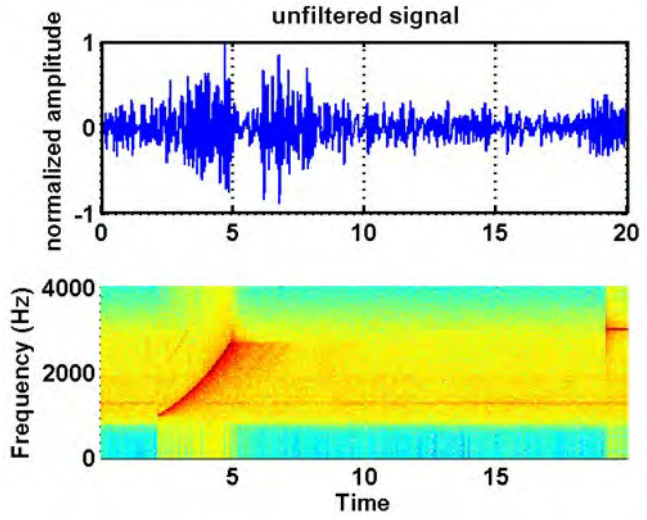

Figure 16. Normalized pressure amplitude and corresponding frequency of an up-sweep signal; visualized as rising frequency over time in the second plot

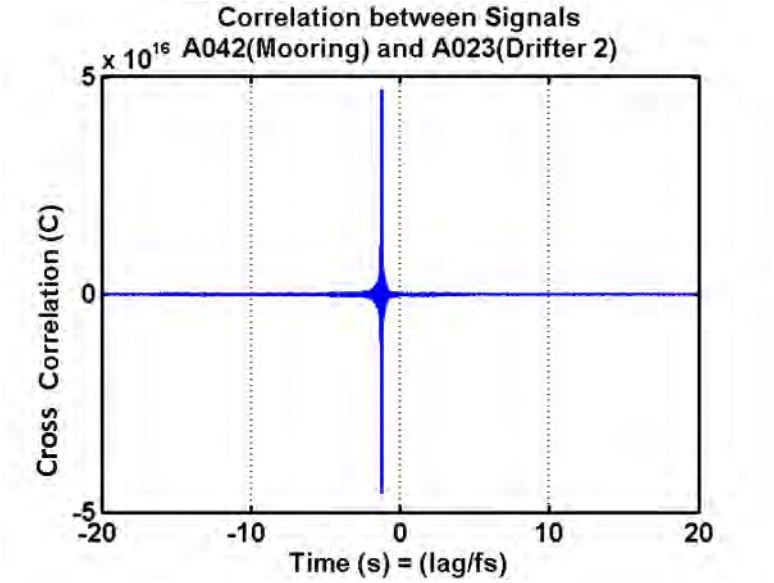

Figure 17. Cross-correlation of the received pressure measurement of two underwater acoustic sensors, A042 and A020, representing the time difference of arrival of ~1.75 sec.

It is worth noting that discrimination of like signals received by two sensors is greatly improved with the use of a sweep, where the change in frequency is very apparent. The use of other signals can make the time-delay estimation much more complex.

THIS PAGE INTENTIONALLY LEFT BLANK

III. TRACKING METHODS FOR SUBMERGED TARGETS

Consider the AquaQuad scenario, where four sensors are using a single measurement type to track a submarine. Like all sensors, the measurements obtained by the AquaQuads will be corrupted by noise and often bias as well. Errors also exist when predicting the future state of the submarine, in what is formally referred to as the process model but can be understood in this case as a dead reckoning solution for submarine motion. If the system of equations that relate the measurements to the position of the target were to be solved analytically, these errors would not be accounted for, and the accuracy of the uncorrected position estimate would suffer accordingly. This is a common estimation problem, and the broad solution to it can be found in Kalman filtering architecture.

A. KALMAN FILTER BASICS

The Kalman filter (KF) is used to recursively estimate a set of states from a linear system by weighting the difference between a measurement and its expected value. The algorithm seeks optimality by minimizing a covariance matrix describing the error of the state estimate. Many derivations and examples of linear Kalman filters exist, so this thesis will not delve into them further. The interested reader should consult [16] for an excellent treatment of the KF and its subsequent variations. There are many symbols and terms that are common to all Kalman filters, so we will describe these components next in their discrete time form. The KF is often derived in continuous time and applied in discrete time. This is because Kalman filters are usually derived and analyzed in continuous form and then implemented on a digital computer, where a discrete time step is required [16].

First, we assume we have a vector x that describes the state of our system. In this thesis, the state vector being estimated is

$$x = \begin{bmatrix} X & Y & u & v & a_x & a_y \end{bmatrix}^T \tag{III.1}$$

which simply contains the 2 dimensional Cartesian plane coordinate position (X,Y), component velocities (u,v) and component accelerations (a_x, a_y). Assuming a standard state-space representation of a linear system, we then have a linear process model that relates the future state of the system x_{k+1} to the current state of the system x_k,

$$x_{k+1} = Ax_k + Bu_k + \omega_k \tag{III.2}$$

where the system matrix A advances the state and is subject to the control effort u_k applied to the states by the input matrix B. The term ω_k is the additive process noise that inevitably corrupts this future state prediction and must be corrected for in our estimation problem.

The measurement vector z_k follows next. In practice, it is populated by measurements from the sensors in the system, but it is fundamentally considered a function of the current state of the system related by the matrix H and once again corrupted by a noise term, v_k.

$$z_k = Hx_k + v_k \tag{III.3}$$

To make the measurement vector more intuitive, consider the scenario where a sensor is able to provide direct measurements of our state variables. In this case, the measurement matrix H is simply the identity matrix. Otherwise, in the linear case, some transformation must occur, and the H matrix is the vehicle for that.

The terms v_k and ω_k are assumed to add Gaussian white noise with zero-mean. Therefore, we use the terms Q for process noise and R for measurement noise to represent their covariance in the KF, which is obtained using the expected value operator, "E" as follows.

$$Q = E[\omega_k \omega_k{}^T] = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xu}^2 & \sigma_{xv}^2 & \sigma_{xa_x}^2 & \sigma_{xa_y}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yu}^2 & \sigma_{yv}^2 & \sigma_{ya_x}^2 & \sigma_{ya_y}^2 \\ \sigma_{ux}^2 & \sigma_{uy}^2 & \sigma_{uu}^2 & \sigma_{uv}^2 & \sigma_{ua_x}^2 & \sigma_{ua_y}^2 \\ \sigma_{vx}^2 & \sigma_{vy}^2 & \sigma_{vu}^2 & \sigma_{vv}^2 & \sigma_{va_x}^2 & \sigma_{va_y}^2 \\ \sigma_{a_x x}^2 & \sigma_{a_x y}^2 & \sigma_{a_x u}^2 & \sigma_{a_x v}^2 & \sigma_{a_x a_x}^2 & \sigma_{a_x a_y}^2 \\ \sigma_{a_y x}^2 & \sigma_{a_y y}^2 & \sigma_{a_y u}^2 & \sigma_{a_y v}^2 & \sigma_{a_y a_x}^2 & \sigma_{a_y a_y}^2 \end{bmatrix} \tag{III.4}$$

$$R = E[v_k v_k{}^T] = \begin{bmatrix} \sigma_{11}^2 & \sigma_{21}^2 \\ \sigma_{12}^2 & \sigma_{22}^2 \end{bmatrix} \tag{III.5}$$

Note that cross-covariance terms populate the off-diagonal elements, and elements in the *R* matrix are left as generic entries to facilitate measurement type. The *Q* and *R* matrices are common "tuning knobs" for the Kalman filter, and their values greatly affect its performance. The R matrix represents the uncertainty in the measurement and can be initialized using values from a sensor's specification sheet (e.g., a specific acoustic USBL sensor has a bearing accuracy of +/- 2.5 degrees, therefore its entry is 2.5^o, converted to radians and squared). Elements of the *Q* matrix are more complex to determine directly, so its entries are often adjusted empirically [17].

Next, we consider the KF covariance matrix *P*. This matrix represents the error of the filter's state estimation,

$$e_k = x_k - \hat{x}_k \tag{III.6}$$

which is composed of the difference between the true state x_k (unknown) and the estimated state $\hat{x}_k$ and whose covariance is computed as

$$P = E[e_k e_k^T] = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xu}^2 & \sigma_{xv}^2 & \sigma_{xa_x}^2 & \sigma_{xa_y}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yu}^2 & \sigma_{yv}^2 & \sigma_{ya_x}^2 & \sigma_{ya_y}^2 \\ \sigma_{ux}^2 & \sigma_{uy}^2 & \sigma_{uu}^2 & \sigma_{uv}^2 & \sigma_{ua_x}^2 & \sigma_{ua_y}^2 \\ \sigma_{vx}^2 & \sigma_{vy}^2 & \sigma_{vu}^2 & \sigma_{vv}^2 & \sigma_{va_x}^2 & \sigma_{va_y}^2 \\ \sigma_{a_x x}^2 & \sigma_{a_x y}^2 & \sigma_{a_x u}^2 & \sigma_{a_x v}^2 & \sigma_{a_x a_x}^2 & \sigma_{a_x a_y}^2 \\ \sigma_{a_y x}^2 & \sigma_{a_y y}^2 & \sigma_{a_y u}^2 & \sigma_{a_y v}^2 & \sigma_{a_y a_x}^2 & \sigma_{a_y a_y}^2 \end{bmatrix} \quad \text{(III.7)}$$

which shares a similar structure with the Q matrix in Equation (III.4). The difference lies in the fact that Q represents the noise inherent to predicting the future state of the system, whereas P represents the error in the filter's estimate of the current state of the system. The P matrix is also fundamental to the KF in that the algorithm actively works to minimize its trace [16], hence minimizing the error in the estimate of the entire state.

Lastly, we discuss the Kalman gain variable K_k. The Kalman gain is that value which weights the correction of your predicted state estimate $\hat{x}_k^{'}$ (a priori) in order to obtain a new estimate $\hat{x}_k$ (a posteriori). The gain is applied to the difference between a measurement z_k and a predicted measurement $H\hat{x}_k^{'}$ via the equation below.

$$\hat{x}_k = \hat{x}_k' + K_k(z_k - H\hat{x}_k') \quad \text{(III.8)}$$

The Kalman gain is calculated as a function of the filter covariance, P_k and measurement noise R. Therefore when the covariance in the estimate is large, the Kalman gain will also be large, and the system will tend to pay more attention to the measurements [16].

$$K_k = P_k' H^T (HP_k' H^T + R)^{-1} \quad \text{(III.9)}$$

All of these equations are used in a cyclic manner, the generic prediction-correction structure of which is shown in Figure 18.

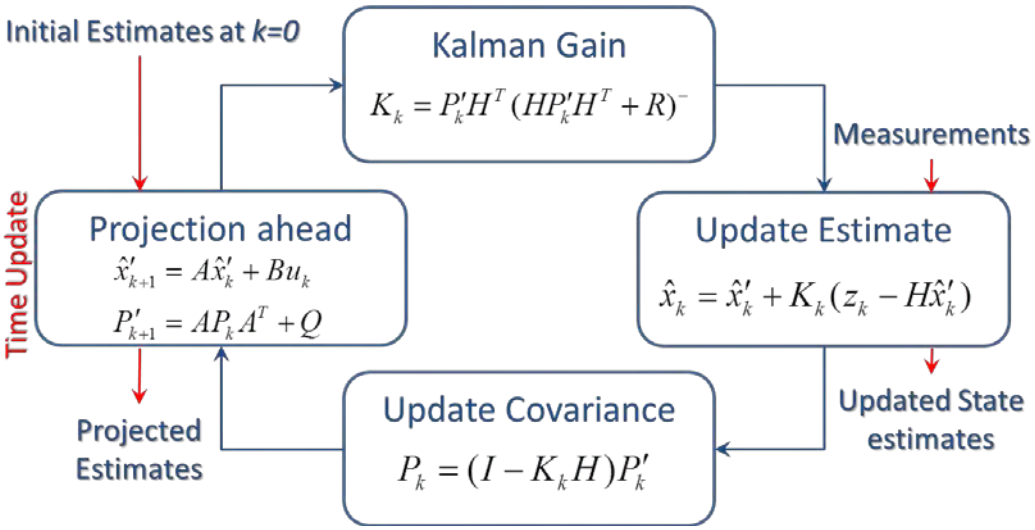

Figure 18. Cyclic process of the Kalman filter, with its primary equations shown, after [17]

Beginning with an initial estimate (assumption at zero time) of the state $\hat{x}_k$ and each covariance matrix *P*, *Q*, and *R*, the future state $\hat{x}'_{k+1}$ and future covariance P'_{k+1} are predicted in the Projection step and are so annotated with a prime designation. Following this step, a subscript notation change occurs, since the "*k+1*" projection is now considered time "*k*." The Kalman gain is then calculated based upon the covariance in the filter's estimate P'_k and the covariance in the measurement *R*. In the update step, a measurement z_k is ingested and compared to the expected measurement $H\hat{x}'_k$. The Kalman gain weights this comparison and adds it to the predicted future state $\hat{x}'_k$ to make the new filter estimate $\hat{x}_k$. Lastly, the covariance matrix P_k is updated, and the cycle begins anew.

B. FORMULATION OF THE NONLINEAR ESTIMATION PROCESS

Several assumptions go into a Kalman filter's use, but a common one is that the noise that drives the process and measurement equations is Gaussian. It should be noted that [16] states, "The Kalman filter is the optimal estimator when the noise is Gaussian, and it is the optimal *linear* filter when the noise is not Gaussian." Therefore if a linear system has non-Gaussian noise, the Kalman filter is still the optimal choice for

estimation. That being said, we consider the case of Gaussian white noise and focus our discussion on how nonlinearity affects this.

It is assumed that the state estimate is a Gaussian random variable (GRV) and so has a defined first and second moment [18]. This is beneficial since these two moments take a trivial amount of space to store but can be used to describe the system in complex ways [19]. The linearity of the system (process and the measurement models) ensures that the Gaussian nature of the random variables is maintained.

When the process or measurement equations are not linear, the optimality of the KF is no longer guaranteed, and the Gaussian property of the noise propagated through a nonlinear equation is also no longer guaranteed. In the specific case of bearings-only tracking, nonlinearity exists in the transformation of bearing measurements into their corresponding position estimate in Cartesian coordinates. This violates one of the basic tenants of the KF since after being operated on by the nonlinear measurement equation the Gaussian nature of the random variables describing the state vector is skewed [1]. To illustrate this phenomenon, [19] conducted a Monte-Carlo simulation with a hypothetical sensor that measures the range and bearing of an object. The measurement equation is the basic polar to Cartesian transformation,

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \begin{bmatrix} r\cos\theta \\ r\sin\theta \end{bmatrix} \tag{III.10}$$

which is inherently nonlinear. The true range and bearing value of a target located at a position of (0,1) was artificially "sampled" several hundred times, with additive zero-mean Gaussian noise included in each sample of r and θ. When these corrupted measurements were transformed back into their Cartesian representation and plotted, Figure 19 from [19] was the result. Note that instead of a circular Gaussian distribution around the target position, the distribution follows a crescent shaped arc, corresponding to a band of range error and a spread of bearing error [19].

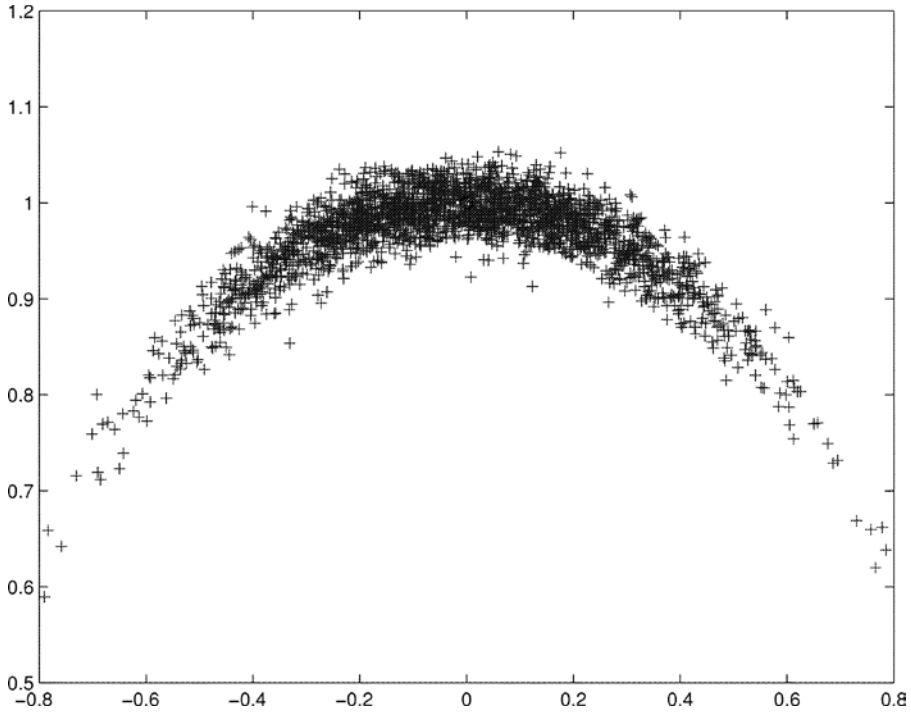

Figure 19. Monte Carlo distribution of the polar-to-Cartesian transformation of a target at position (0,1) with zero-mean Gaussian measurements, from [19]

This type of nonlinearity occurs very commonly in dynamical systems. In order to overcome this obstacle, yet maintain the elegant structure and power of the KF, extensions to the algorithm have been created. This thesis investigates two of them with the goal of developing objective comparison data and determining which is the most suitable for use on the AquaQuad platform.

C. EXTENDED KALMAN FILTER

The extended Kalman filter (EKF) is a widely used algorithm for nonlinear estimation [20]. It mitigates nonlinearities in the process and measurement equations through their linearization. Conceptually, this linearization is conducted to the "first-order" using the first two terms of the Taylor series expansion, shown for a generic function *f(x).*

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \ldots \quad \text{(III.11)}$$

The implied assumption is that the linearized statistical properties of the random variable being estimated are approximately equal to its nonlinear properties [16]. This approximation breaks down for highly nonlinear systems and can lead to large errors in the posterior mean and covariance estimates or general filter divergence [20]. Within the EKF equations, linearization is conducted utilizing a Jacobian matrix of partial derivatives, the use of which creates its own problems: the Jacobian can often be difficult to calculate, the reliability of calculating derivatives requires very small sampling time, the derivatives of noisy signals amplify the power of noise, and there are also some instances where it does not exist [19]. These limitations set aside, the performance of the EKF remains acceptable and often excellent for the majority of applications, leading to its near-ubiquitous use.

The structure of the EKF is very similar to KF and proceeds through the prediction-correction update process of Figure 20 after [17], which can be compared to the KF case in Figure 18.

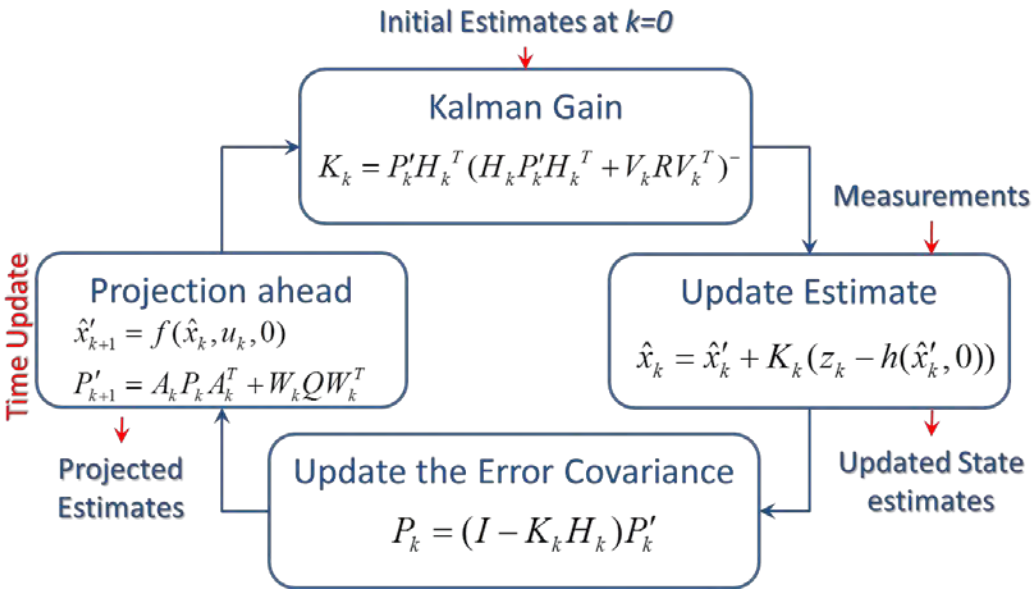

Figure 20. Cyclic process of the Extended Kalman filter, with its primary equations shown, after [17]

The key difference between Figure 18 and Figure 20 is rooted within the nonlinear state projection $\hat{x}_{k+1} = f(\hat{x}_k, u_k, 0)$ and measurement update $h(\hat{x}_k^{'}, 0)$. These equations force the creation of linearizing Jacobians for the terms A_k and W_k in the "Projection ahead" step, H_k in the "Kalman gain" and "Update the Error Covariance" steps, and V_k in the "Kalman gain" step. The generic structure of this matrix of partial derivatives follows, from [17].

$$A_{ij} = \left.\frac{\delta f_i}{\delta x_j}\right|_{(\hat{x}_k, u_{k,}\, 0)}$$

$$W_{ij} = \left.\frac{\delta f_i}{\delta w_j}\right|_{(\hat{x}_k, u_{k,}\, 0)}$$

$$H_{ij} = \left.\frac{\delta h_i}{\delta x_j}\right|_{(\hat{x}_k, 0)}$$

$$V_{ij} = \left.\frac{\delta h_i}{\delta v_j}\right|_{(\hat{x}_k, 0)} \qquad \text{(III.12)}$$

A number of variations to the EKF exist, however, the version utilized in this thesis considers the case of additive noise in the process and measurement model and otherwise follows a format similar to that in [17]. The specific equations used for a bearing-only measurement tracking scenario are those presented in Figure 21, where the state being estimated consists of the target position (x_o, y_o), component velocity (u_o, v_o) and component acceleration (ax_o ay_o). Note the use of the Φ term in the process equation. This is a common and convenient way of updating the state based upon a first-order linear differential equation $\dot{x} = Ax$ whose solution is $\Phi = e^{A\Delta t}$.

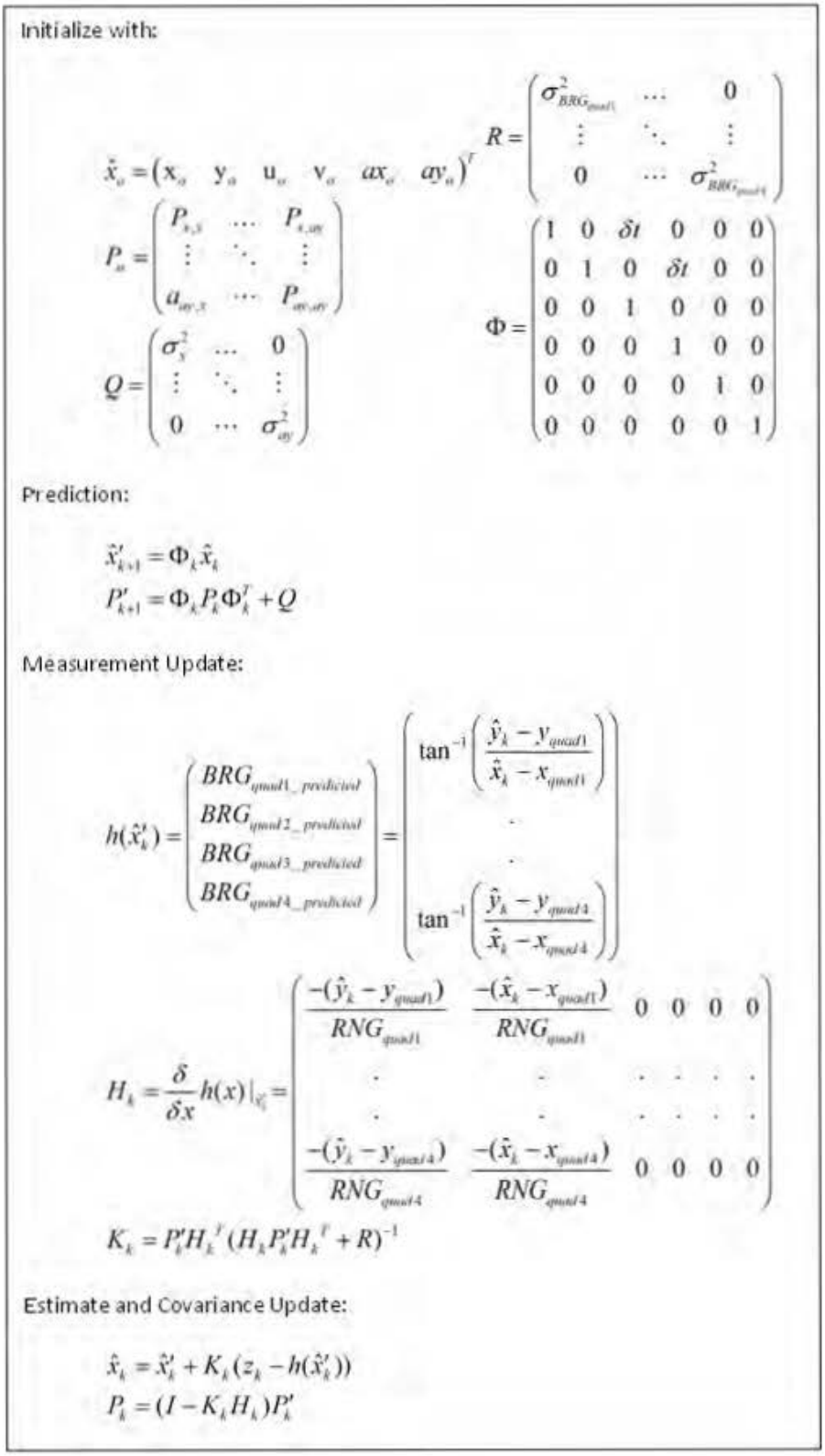

Figure 21. Extended Kalman filter (EKF) algorithm utilized for bearing-only measurement tracking simulation, after [17]

D. UNSCENTED KALMAN FILTER

In order to overcome some of the previously stated limitations of the EKF, the unscented Kalman filter (UKF) was proposed. The UKF omits the linearization step and uses the true nonlinear functions that describe the process and measurement equations. This greatly simplifies the complexities that arise when calculating the Jacobian matrices at each iteration of the filter. The UKF was created on the basis that it is simpler to transform a single point than an entire probability distribution and that the desired probability distribution can then be reconstructed after an unscented transformation [16]. The states are still represented as Gaussian random variables, but their normal distribution is maintained by using a set of specifically designed sample points or "sigma points" around the estimate in an unscented transformation (UT) [20].

The UT and its sigma points are fundamental to the UKF since they allow us to create an approximate probability distribution that has not been skewed by nonlinear functions [19]. The sigma points and the weights associated with them must be chosen carefully such that the ensemble mean and covariance is a good estimate of the true mean and covariance [16], and in this case they are selected such that they create a Gaussian distribution. Figure 22 from [20] compares the effect of this transformation on the resultant mean and covariance (far right) with that generated from a Monte Carlo sampling routine (far left), and EKF linearization (center). It is clear that the UT provides a much closer estimate of the true distribution and does so using relatively few sigma points in this case.

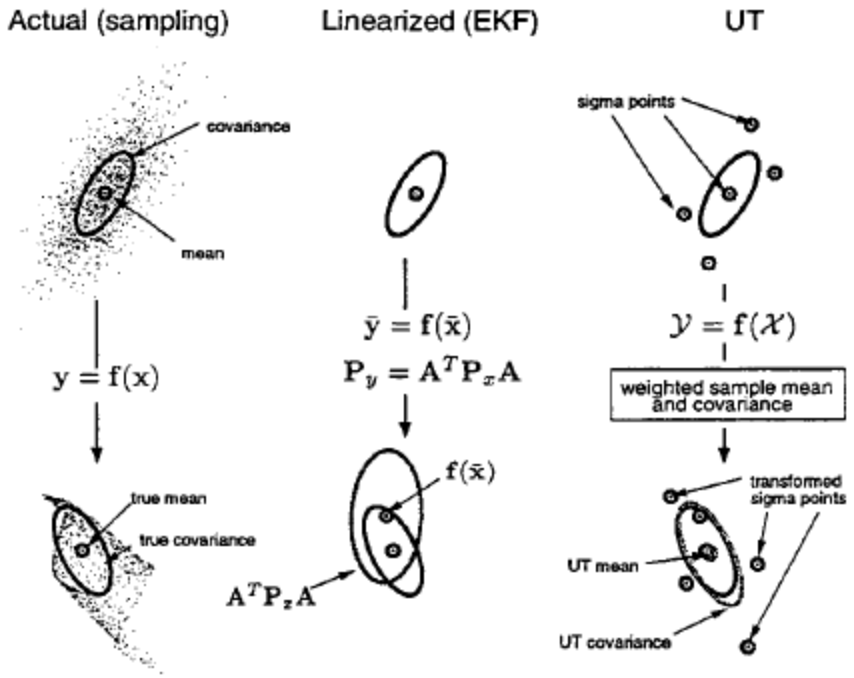

Figure 22. Mean and covariance propagation for Monte Carlo sampling, EKF linearization, and the unscented transformation, from [20]

The UKF has several advantages over the EKF, and the authors of [20] define that it has a similar computational cost that is within one order of magnitude of the linearized filter. Moreover, the functions with discontinuities for which there is no partial derivative can be easily spanned by the sigma points, thereby mitigating the EKF requirement of Jacobian existence at every step [19]. The higher order terms of the Taylor series expansion in Equation (III.11), which are neglected when using the EKF, are also more accurately represented in the UKF. Posterior mean and covariance are reportedly captured in statistical moments up to the third order [20]. From a practical standpoint, the user can directly substitute the nonlinear process and measurement equations into the UKF, with no need to calculate the complicated matrices of partial derivatives that the EKF requires. This makes the UKF a very attractive filter.

Once again, the equations utilized in this thesis have been adapted following the structure seen in [20], revised for the case of additive noise, presented for the bearing-only measurement tracking scenario and summarized in Figure 23. The MATLAB implementation can be seen in Appendix B.

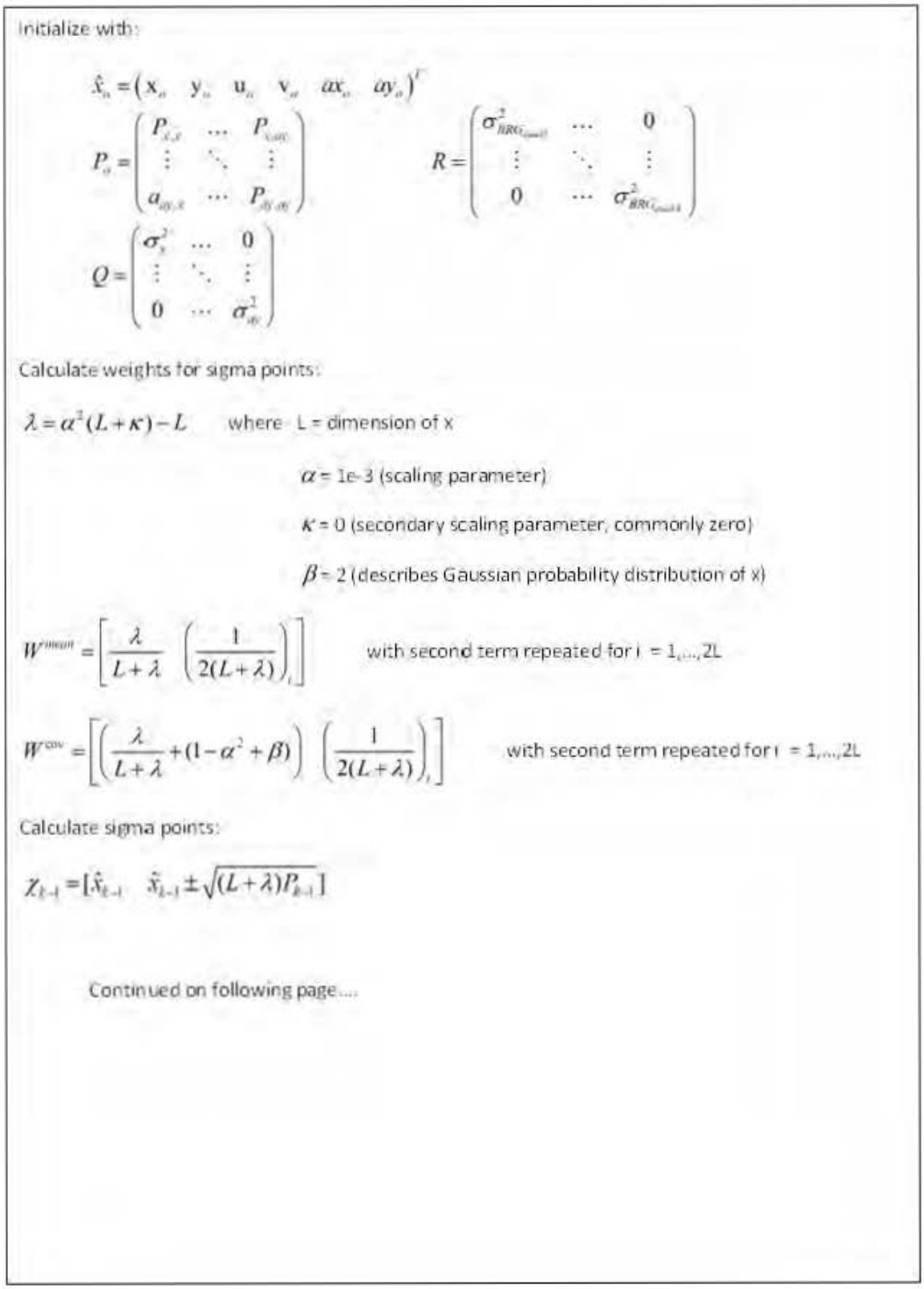

Figure 23. Unscented Kalman filter (UKF) algorithm utilized for bearing-only measurement tracking simulation, after [20].

Unscented Transformation of Process Equations:

$$\chi_{k|k-1} = F(\chi_{k-1}) = \begin{pmatrix} 1 & 0 & \delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & \delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \chi_{k-1}$$

$$\hat{x}_k^- = \sum_{i=0}^{2L} W_i^{mean} \chi_{i,k|k-1}$$

$$P_k^- = (\chi_{i,k|k-1} - \hat{x}_k^-) diag(W^{cov}) (\chi_{i,k|k-1} - \hat{x}_k^-)^T + Q$$

Unscented Transformation of Measurement Equations:

$$\hat{Y}_{k|k-1} = H(\chi_{k-1}) = \begin{pmatrix} \tan^{-1}\left(\dfrac{\chi_{k-1}(2) - y_{quad1}}{\chi_{k-1}(1) - x_{quad1}}\right) \\ \vdots \\ \tan^{-1}\left(\dfrac{\chi_{k-1}(2) - y_{quad4}}{\chi_{k-1}(1) - x_{quad4}}\right) \end{pmatrix}$$

$$\hat{y}_k^- = \sum_{i=0}^{2L} W_i^{mean} Y_{i,k|k-1}$$

$$P_{\tilde{y}_k \tilde{y}_k} = (Y_{k|k-1} - \hat{y}_k^-) diag(W^{cov}) (Y_{k|k-1} - \hat{y}_k^-)^T + R$$

Update Equations:

$$P_{\tilde{x}_k \tilde{y}_k} = (\chi_{i,k|k-1} - \hat{x}_k^-) diag(W^{cov}) (Y_{k|k-1} - \hat{y}_k^-)^T$$

$$K = P_{\tilde{x}_k \tilde{y}_k} P_{\tilde{y}_k \tilde{y}_k}^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K(z_k - \hat{y}_k^-)$$

$$P_k = P_k^- - K P_{\tilde{x}_k \tilde{y}_k}^T$$

Figure 23 (cont'd) Unscented Kalman filter (UKF) algorithm utilized for bearing-only measurement tracking simulation, after [20].

E. COMPARISON OF EXTENDED AND UNSCENTED KALMAN FILTER PERFORMANCE

In order to evaluate filter performance, both the EKF and UKF were used in simulation, facilitating the comparison of mean square error, computation time, and convergence properties between the two filters. The simulation was conducted in SIMULINK under the devised scenario of four AquaQuads tracking a submerged contact with bearing-only measurements. Each AquaQuad and the target submarine were modeled as point masses in SIMULINK. Bearing measurements were determined by simple trigonometry using the known sensing node (quadrotor) and submarine positions. Band-limited white noise was added to these measurements with a power of $1x10^{-4}$ and seed numbers that were independent from one another. The SIMULINK model used in the UKF simulation, closely matched by its EKF counterpart, is shown in Figure 24, showing the general setup and relevant signal routing.

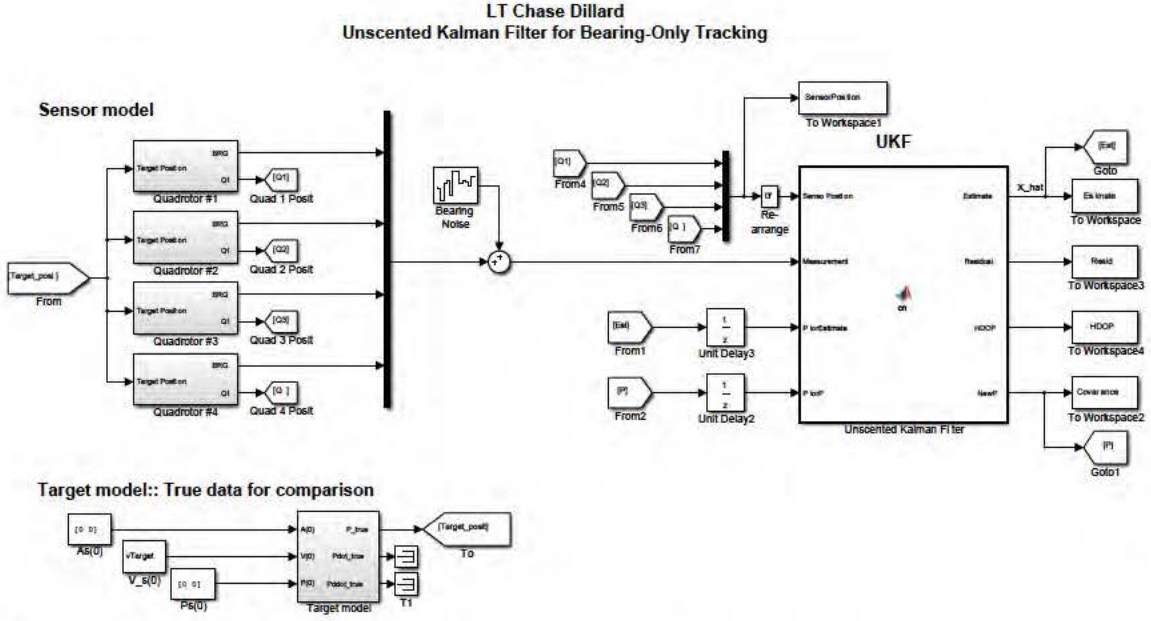

Figure 24. SIMULINK model for bearing-only tracking simulation, using an unscented Kalman filter and point-mass kinematics

Figure 25 displays the paths taken by the sensors and targets within the simulation, without the position estimate from either filter. The AquaQuads form a

perimeter around the submarine and drift in a circular pattern at speeds substantially less than that of their quarry. The target, meanwhile, originates at the center of the group and proceeds continuously outward. This changes the relative geometry of the scenario, which has impacts that will be considered in following chapters.

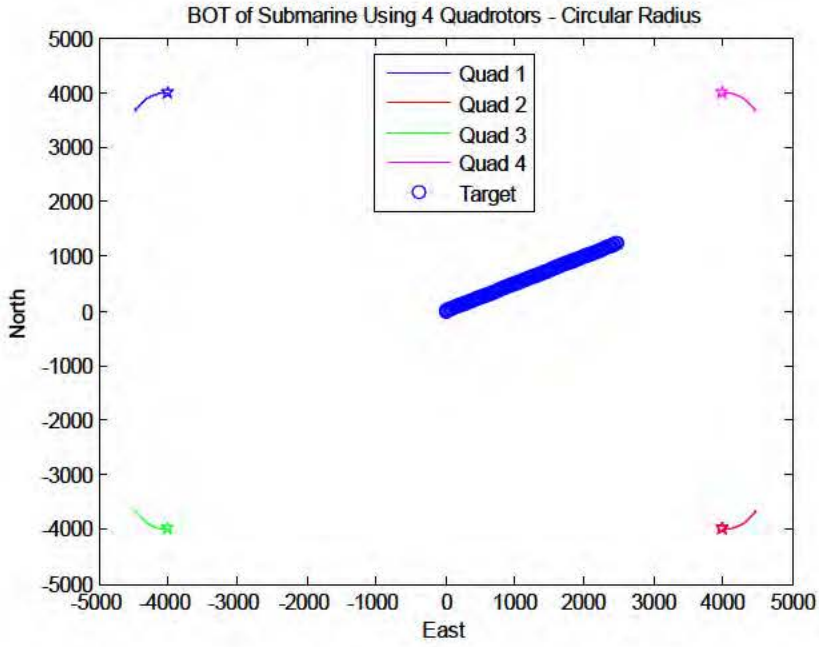

Figure 25. Relative positions of four AquaQuads and a target submarine in SIMULINK simulation

Both the EKF and UKF were evaluated under this described scenario, and each was initialized with the same starting parameters, namely:

- Initial estimate, $\hat{x}_0 = \begin{bmatrix} X_o \\ Y_o \\ u_0 \\ v_0 \\ a_{x0} \\ a_{y0} \end{bmatrix} = \begin{bmatrix} 100 \\ 100 \\ 20 \\ 20 \\ 0 \\ 0 \end{bmatrix}$ (III.13)

- Process noise, $Q = diag([10e^{-6} \quad 10e^{-6} \quad 1e^{-6} \quad 1e^{-6} \quad 1e^{-6} \quad 1e^{-6}])$ (III.14)

- Measurement noise, $R = diag([1e^{-6} \quad 1e^{-6} \quad 1e^{-6} \quad 1e^{-6}])$ (III.15)

Additionally, the CPU time required to run each scenario was determined using the *tic* and *toc* commands in MATLAB. These commands start and stop an internal timer at the beginning and end of each simulation. This provides a metric for comparison of which filter is more computationally expensive to use.

Figure 26 displays a reduced-scale plot of the EKF's estimated submarine position laid over top of its actual track. At first glance, it appears to be a good estimate, and it certainly adheres to the track well. The elapsed time to run the EKF in this scenario was 0.851532 seconds.

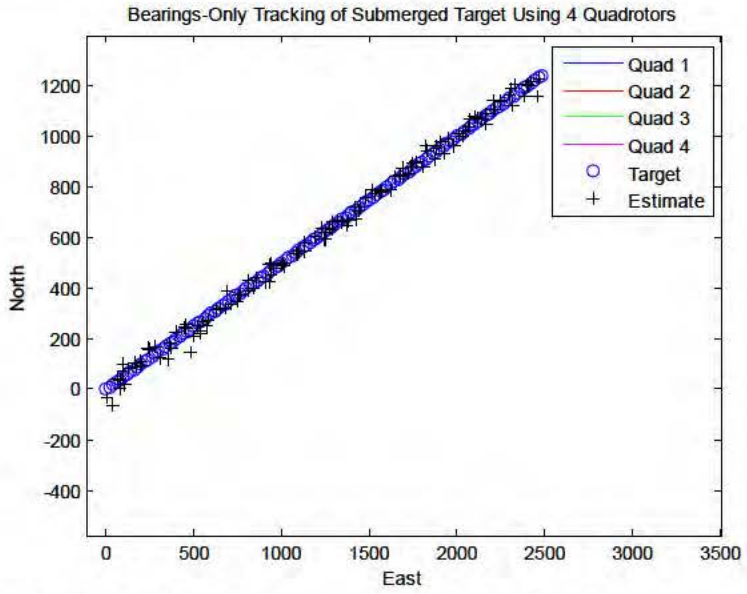

Figure 26. Estimated submarine position using bearing-only measurements and an extended Kalman filter

A closer look at the plot of residuals (difference between known and estimated position) in Figure 27 shows a position estimate that never truly converges. The velocity residual approaches zero at approximately 150 seconds.

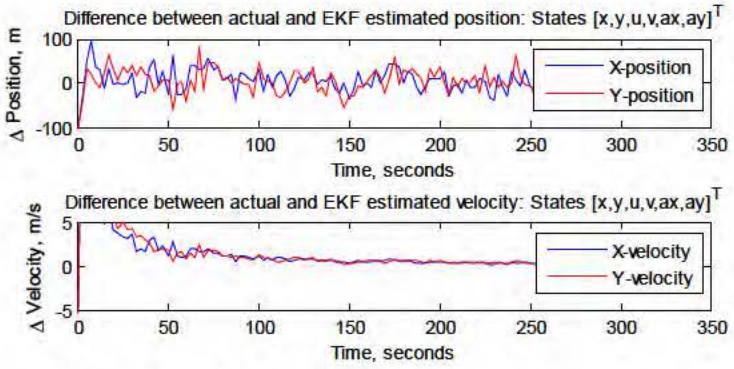

Figure 27. EKF residuals in a bearing-only measurement tracking scenario

These figures can be directly compared to those generated using the UKF. The UKF implementation shares many of the same characteristics of the EKF. The filter estimation overlay of Figure 28 also appears to track the true position of the submarine well. Elapsed time for the UKF implementation was 1.076891 seconds, which is on par with the EKF.

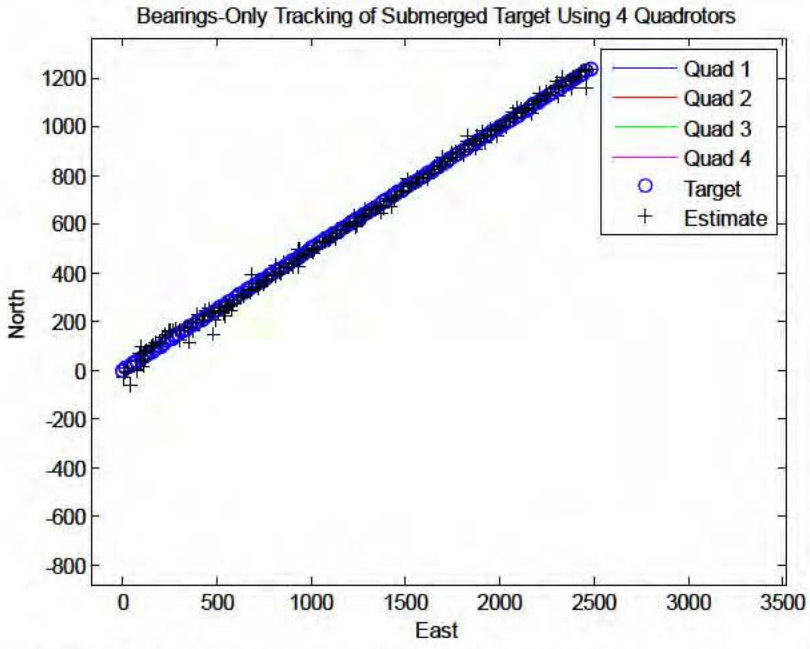

Figure 28. Estimated submarine position using bearing-only measurements and an unscented Kalman filter

Once again, the residual plot displays the truth of the UKF's state estimate. Compared to Figure 27, the UKF residuals in Figure 29 are significantly smoother, and there is position and velocity convergence within approximately 75 seconds. This property of smooth UKF convergence was seen in nearly every comparison run using the two filters and appears to speak to the superiority of the UKF.

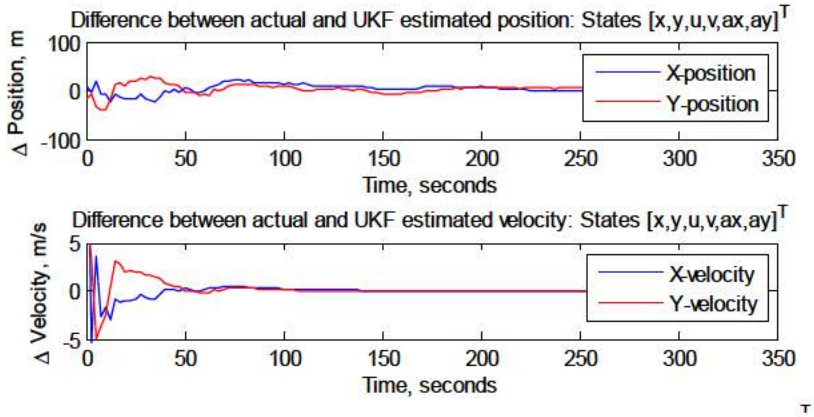

Figure 29. UKF residuals in a bearing-only measurement tracking scenario

In order to analyze the results from Figure 27 and Figure 29 more objectively, Table 2 displays the mean square error of the position and velocity state estimates from each filter in this scenario. The final column of Table 2 represents the difference, or "delta", in mean square error when comparing the EKF and UKF simulations. A positive delta value indicates superiority in the UKF estimate. The term "circular perimeter" in the table heading of Table 2 was used to differentiate the starting geometry of the AquaQuads with other scenarios that were tested. Those scenarios will not be repeated here for brevity, but the results were similar: under the precepts of the described simulation, the UKF routinely possessed a greatly reduced mean square error and convergence time, as compared to the EKF.

		EKF Circular Perimeter	UKF Circular Perimeter	Circular Perimeter Delta (EKF-UKF)
Mean Square Error	X, $(m)^2$	*765.7431*	*97.6823*	668.0608
	Y, $(m)^2$	*719.7339*	*116.8798*	602.8541
	U, $(m/s)^2$	*19.7663*	*12.4691*	7.2972
	V, $(m/s)^2$	*13.7761*	*10.2321*	3.544

Table 2. Comparison of the mean square error of an EKF and a UKF, using bearing-only measurements in a tracking scenario

IV. ENERGY-EFFICIENT PERSISTENT SURVEILLANCE

With our estimation filter designated, we proceed on towards the other main component of this thesis: developing a process facilitating persistence of surveillance. At its fundamental level, this process amounts to path planning and flock coordination. It is important to note the two major points of emphasis.

First is the energy management component, which can be controlled with path planning. Planning the motion of a robot is a fundamental problem that has been tackled in many ways and one of them is with the use of sampling-based algorithms. These algorithms have a proven ability to quickly explore a given space and develop a path between the initial and goal state that is free of obstructions. To do so, they randomly sample points in the state space and attempt to make obstacle-free connections to them from existing points. This sampling behavior leads to probabilistic completeness: as the number of iterations increases, the probability of finding a successful path to the goal approaches one [21]. There are many variations of these algorithms, and the concept is extremely flexible, allowing us to produce a desired behavior in the paths generated.

For our purposes, we not only seek an obstacle-free path to a goal state, we also desire optimality in the energy that is expended to achieve it. The control authority available to the AquaQuad platform is also very limited, so we must address the unique combination of drifting (dead-reckoning) and flight periods in the planning process. To these ends, we created a new algorithm specific to the AquaQuad scenario based upon rapidly-exploring random tree (RRT) [22] that we call dead-reckoning rapidly-exploring random tree star (DR-RRT*).

While increasing the on-station time of the AquaQuads is integral to their task as autonomous systems, it cannot come at the expense of their primary mission to track the target, and therefore this is the other point of emphasis. The authors sought a metric that could be used to define the quality of the group’s ability to estimate the position of the target in different geometric configurations. This provides an input to the path planning

process and also contributes a threshold that dictates when repositioning should occur, requiring flock coordination. We found this metric in the geometric dilution of precision.

A. DILUTION OF PRECISION AS AN OPTIMIZATION METRIC

The concept of dilution of precision (DOP) comes from classical GPS theory, where it is used as a metric for the quality of a position fix. The number of satellites and their relative geometry around a receiver play the dominant role in the calculation of this quantity. The GPS pseudorange measurement in the following discussion is directly analogous to the underwater range-only measurement for our tracking scenario, but it will be shown that the nature of DOP is relatable using any measurement.

As a simple example, consider the two dimensional (XY plane) case of two satellites positioned such that the receiver is between them. The position of the receiver with respect to one of those satellites can lie anywhere on the circumference of a circle whose radius is defined using the difference between the sent and received times, multiplied by the speed of light. With two satellites, an additional circle is generated, and the intersection points of these circles indicate two possible locations of the receiver. Three satellites would theoretically eliminate one of those possible locations, leaving the user with a single estimated (x,y) position. Consider, however, that many errors exist in determining the range between satellite and receiver. One of these major sources of error is the difference between the clocks on board the satellite and receiver. If they were to be off from one another by a single second, that positional error would be on the order of 10^8 meters based upon the speed of light. Therefore the true range from satellite to receiver lies inside a bounded region of possible circles, dependent upon the magnitude of the error. Relative geometry can help shrink this region of uncertainty, as Figure 30 from [18] portrays very well.

Figure 30. GPS pseudoranges and their associated area of uncertainty under different geometric satellite configurations, from [18]

Delving deeper into the GPS equations from [18] provides the motivation for using dilution of precision in this work. Each pseudorange measurement can be modeled per Equation (IV.1).

$$\tilde{\rho}^i = \left\| \mathbf{p} - \mathbf{p}^i \right\| + c\Delta t_r + c\delta t^i + \frac{f_2}{f_1} I_r^i + T_r^i + M_{\rho_1}^i + v_{\rho_i}^i \tag{IV.1}$$

The first term in Equation (IV.1) is the norm of the difference between the receiver position **p** and the satellite position $\mathbf{p}^i$, which in other words is the true (unknown) range that separates them. The remaining portions of the equation are all error effects that corrupt the measurement of this true range. Specifically, the next two terms both multiply the speed of light by the clock errors between source and receiver. The first clock error $c\Delta t_r$ is one that we can apply corrections for. The second clock error $c\delta t^i$ is that residual portion that we can never precisely provide a correction for in our estimation process. The last terms correspond to dispersive atmospheric effects, non-dispersive atmospheric effects, multipath errors, and measurement noise, respectively. Chapter 8 in [18] discusses these error terms in great detail. All error components are considered statistically independent, and the square root of their sum of squares is designated the user equivalent range error (UERE).

Continuing our abbreviated derivation of DOP, [18] provides an iterative algorithm to minimize the error in the estimation of $\hat{x} = \left[\hat{X} \quad \hat{Y} \quad \hat{Z} \quad c\Delta\hat{t} \right]$, seen in Equation (IV.2).

$$\hat{x}_{k+1} = \hat{x}_k + (H^T H)^{-1} H^T (\tilde{\rho} - \hat{\rho}(\hat{x}_k)) \tag{IV.2}$$

where

$$H = \left[\begin{array}{c} \frac{\delta \rho^1}{\delta x} \\ \frac{\delta \rho^2}{\delta x} \\ \frac{\delta \rho^3}{\delta x} \\ \frac{\delta \rho^4}{\delta x} \end{array} \right]_{\hat{x}_k} = \left[\begin{array}{cc} \frac{\hat{p}_k - \hat{p}^1}{\left\| \hat{p}_k - \hat{p}^1 \right\|} & 1 \\ \frac{\hat{p}_k - \hat{p}^2}{\left\| \hat{p}_k - \hat{p}^2 \right\|} & 1 \\ \frac{\hat{p}_k - \hat{p}^3}{\left\| \hat{p}_k - \hat{p}^3 \right\|} & 1 \\ \frac{\hat{p}_k - \hat{p}^4}{\left\| \hat{p}_k - \hat{p}^4 \right\|} & 1 \end{array} \right] \tag{IV.3}$$

is the Jacobian matrix representing the partial derivative of the measurement equation $\rho(x, \mathbf{p}^i) = \left\| \mathbf{p} - \mathbf{p}^i \right\| + c\Delta t_r$, evaluated at the algorithm's current estimate. Of note, this is the same form of the H matrix utilized in the EKF of Chapter III.C. When Equation (IV.2) is run to its conclusion, we expect that $\delta x = x_{k+1} - x_k \approx 0$ and therefore $(H^T H)^{-1} H^T (\tilde{\rho} - \hat{\rho}(\hat{x}_k)) = 0$. With some manipulation of related equations and designating the user equivalent range error as the variable χ, we arrive at Equation (IV.4).

$$\delta x = -(H^T H)^{-1} H^T \chi \tag{IV.4}$$

We then determine the covariance of Equation (IV.4) as follows in Equation (IV.5),

$$\begin{aligned} P &= E\left\langle \delta x \delta x^T \right\rangle \\ &= (H^T H)^{-1} \sigma^2 \end{aligned} \tag{IV.5}$$

where σ^2 is the variance of the UERE. Equation (IV.5) shows that the covariance of the GPS pseudorange problem is a product of the geometric relative positioning encapsulated in the H matrix and the collective measurement error found in the UERE variance term. The quantity $(H^T H)^{-1}$ magnifies the UERE in estimating the position of the receiver,

and so we isolate this term and convert it to the more-useful scalar metric known as the geometric dilution of precision (GDOP). As we are principally concerned with the Cartesian coordinate estimation of position (x,y), we consider only these elements and so use the term horizontal dilution of precision (HDOP) seen in Equation (IV.6).

$$HDOP = \sqrt{trace(H^T H)^{-1}} \tag{IV.6}$$

All of the preceding information was presented in the vein of GPS measurements; however the extension to the AquaQuad scenario is extremely natural. In place of satellites, we have four quadrotors. Instead of estimating the position of a receiver, we are estimating the position of a submarine. If we were to use range-only measurements, the speed of sound in water would replace the speed of light in air. Depending upon the measurements being used, the equations must be modified slightly. Fortunately, the concept of HDOP is inherently flexible to our purposes, and the only true alterations are conducted inside the Jacobian matrix, H. The metric still defines the potential precision we can achieve in a particular measurement scenario and is therefore incredibly useful in determining the best and potentially optimal arrangement of our sensors.

1. HDOP for Bearing-Only Measurement Tracking

In order to better utilize HDOP, it is important to understand the fundamental configurations that will minimize it and therefore improve it. It may be surprising to discover that, depending on the measurements being obtained, there are varying relationships between HDOP and range to the target. The authors of [23] provide great insight into this relationship, using the notation of Figure 31 as a basis.

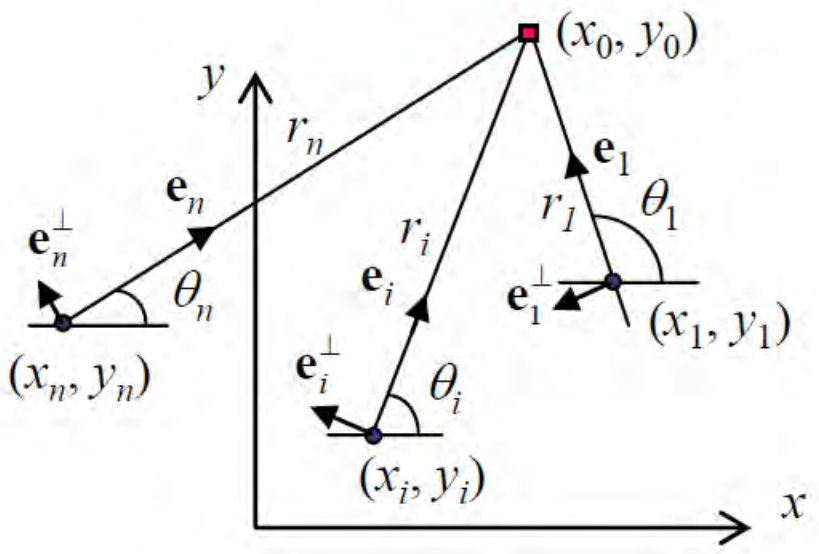

Figure 31. XY plane representation of three sensors taking range and bearing measurements with respect to a target, in red, from [23].

Each sensor at ($x_{1...i...n}$, $y_{1...i...n}$) in Figure 31 is capable of taking a range ($r_{1...i...n}$) or bearing ($\theta_{1...i...n}$) measurement from the target being tracked, which is represented as a red square at position (x_0, y_0). Considering first the case of bearing-only tracking, we begin with the nonlinear measurement equation

$$\theta_i = \tan^{-1}\left(\frac{\hat{y} - y_i}{\hat{x} - x_i}\right) \tag{IV.7}$$

whose Jacobian is represented as

$$H_i = \left[\begin{array}{c} \frac{\delta\theta_i}{\delta x} \\ \frac{\delta\theta_i}{\delta y} \end{array}\right]\Bigg|_{[x_0, y_0]} = \left[\begin{array}{c} \frac{-y_0 - y_i}{r_i^2} \\ \frac{x_0 - x_i}{r_i^2} \end{array}\right] = \left[\begin{array}{c} \frac{-\sin(\theta_i)}{r_i} \\ \frac{\cos(\theta_i)}{r_i} \end{array}\right] \tag{IV.8}$$

with the term $r_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$. Next, [23] considers the form of the HDOP equation that includes the measurement covariance term, *R*, shown in Equation (IV.9). For comparison purposes, the HDOP variant previously presented in Equation (IV.6) considers the specific case where *R* is equal to identity.

$$HDOP = \sqrt{trace((H^T R^{-1} H)^{-1})} \tag{IV.9}$$

After conducting the matrix operations within Equation (IV.9) symbolically, for a scenario with only two sensors and with some trigonometric simplification, we arrive at the bearings-only tracking HDOP equation from [23].

$$HDOP_{BRG-ONLY} = \sqrt{\frac{r_1^2\sigma_1^2 + r_2^2\sigma_2^2}{\sin^2(\theta_1 - \theta_2)}} \tag{IV.10}$$

What is significant about Equation (IV.10) is that a range term shows up explicitly in the numerator of this equation. For the bearings-only tracking scenario, HDOP increases and therefore degrades with increasing range. Also of note is that when the difference in bearings to the target from sensor 1 and sensor 2 (corresponding to θ_1 and θ_2, respectively) are either equal or 180 degrees apart from one another, there is a singularity. Therefore, optimality is found in geometric configurations that place the sensors close to the target and in locations that make right angles to one another.

2. HDOP for Range-Only Measurement Tracking

Next, we consider the range-only tracking scenario whose nonlinear measurement equation is simply r_i from Equation (II.4), which leads to the Jacobian matrix

$$H_i = \left[\begin{matrix} \frac{\delta r_i}{\delta x} \\ \frac{\delta r_i}{\delta y} \end{matrix}\right]_{[x_0, y_0]} = \left[\begin{matrix} \frac{x_0 - x_i}{r_i} \\ \frac{y_0 - y_i}{r_i} \end{matrix}\right] = \left[\begin{matrix} \cos(\theta_i) \\ \sin(\theta_i) \end{matrix}\right] \tag{IV.11}$$

and produces the symbolically simplified HDOP equation from [23].

$$HDOP_{RNG-ONLY} = \sqrt{\frac{\sigma_1^2 + \sigma_2^2}{\sin^2(\theta_1 - \theta_2)}} \tag{IV.12}$$

Once again, we see in Equation (IV.12) that a collinear configuration between two sensors produces a singularity due to the sine term in the denominator. What is different

from the bearing-only scenario is the lack of an explicit range term in the numerator. At first glance, this would lead one to believe that the range-only HDOP equation is not dependent upon the distance between the sensors and receivers, but this is not observed in practice. Due to the simplification conducted in [23] leading to the trigonometric denominator of Equation (IV.12), there is actually a range term buried within it. Therefore, in the case of range-only measurements, as range increases, the HDOP improves.

3. HDOP for TDOA Measurement Tracking

While not covered in [23], we conduct a similar HDOP analysis for the TDOA measurement seen in Equation (II.7). Since the TDOA measurement involves the difference in range (and therefore time) between sensors and the target, a subscript notation change is necessary for the range term r_{iT} or r_{jT} denoting the range between the i^{th} or j^{th} sensor and the target T. The Jacobian for one TDOA measurement between two sensors is seen in Equation (IV.13).

$$H_i = \left[\begin{array}{c} \frac{\delta \tau_{ij}}{\delta x} \\ \frac{\delta \tau_{ij}}{\delta y} \end{array}\right]\Bigg|_{[x_0, y_0]} = \left[\begin{array}{cc} \frac{x_j - x_0}{r_{jT}} & -\frac{x_i - x_0}{r_{iT}} \\ \frac{y_j - y_0}{r_{jT}} & -\frac{y_i - y_0}{r_{iT}} \end{array}\right] = \left[\begin{array}{c} cos(\theta_j) - cos(\theta_i) \\ sin(\theta_j) - sin(\theta_i) \end{array}\right] \tag{IV.13}$$

Rather than determine a closed-form solution for the HDOP equation as seen in Equations (IV.10) and (IV.12) from [23], we present a simulation exploring its behavior that is applicable to our scenario. Specifically, we fix the locations of two sensors and a target while allowing the position of a third sensor to vary over a gridded area. An HDOP calculation is made at every point in the grid and the results plotted. The result of this plot is shown in Figure 32 from both an overhead and a three-dimensional perspective. These views help to illustrate some interesting findings from the analysis. The first is the overhead view (far left Figure 32) in which we see a fairly uniform low value for HDOP, punctuated by singularities (exhibited in white) along the line of bearing that connects QR #1 and QR #2 to the position of the target. The source of these singularities can be traced to the trigonometric representation of Equation (IV.13). When the angle to the

target θ_i is equal to that of θ_j, the corresponding element of H in Equation (IV.13) is zero, therefore the inverse operator of Equation (IV.6) results in a singularity. Consequently, co-aligning TDOA sensors on the same line of bearing is highly undesirable.

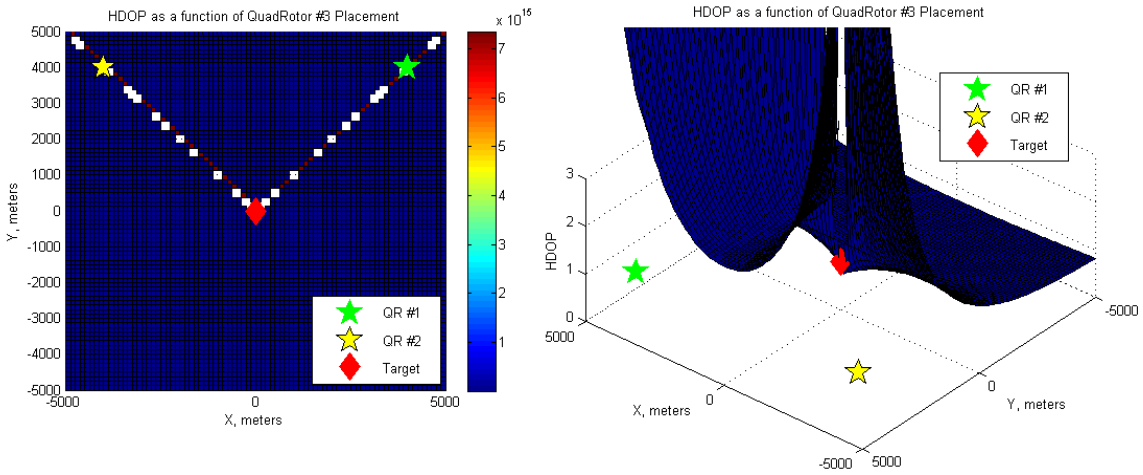

Figure 32. Overhead (left) and three-dimensional view (right) of TDOA HDOP variations due to the movement of a single sensor

The rotated three-dimensional view on the far right of Figure 32 shows that local minimums exist on the Y-axis, however the negative Y-axis is the lesser of the two due to the large angle separation present there between the sensors. When off the Y-axis, the HDOP rises. This is due to the reduced angle differential between θ_i and θ_j.

In our implementation of the TDOA equations we did not observe range dependence, however work has been conducted in [24] and [25] to analyze this aspect. The predominant result of the portion of [24] applicable to our purposes is seen in Figure 33, which displays increasing DOP (degraded precision) with range. The source of this error is described as due to increasing "flatness" of the hyperbolic curves with distance resulting in an intersection point that becomes less definite [25]. Of additional interest in Figure 33 is the reduction in DOP (improved precision) as the number of sensors "n" increases.

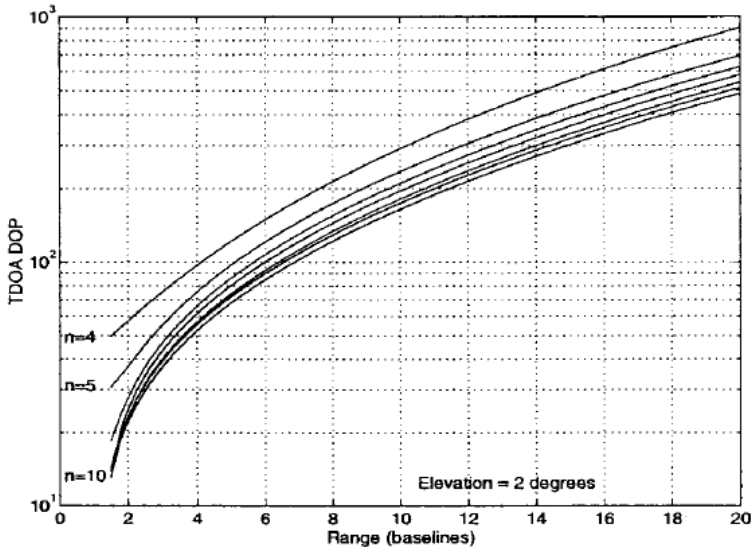

Figure 33. Variation in dilution of precision for the case of TDOA measurements with respect to increasing Range and number of sensors, "n", from [24]

In each of the detailed measurement cases (range-only, bearing-only and TDOA), the dilution of precision provides us a single number that quantifies the potential precision with which we can track the target. This number is based solely on the relative location of each AquaQuad, which is known, and the current position estimate of the target, which is shared. The limited number of inputs and simplicity of the calculation leads to its repeatability and allows us to intelligently plan the positioning of the AquaQuads for tracking.

4. Optimal Sensor Placement for HDOP

A logical extension to the preceding discussion of geometry-based HDOP is answering the question of which sensor configurations minimize the metric. To approach this, we explored the results that were obtained when minimizing the bearing-only and range-only HDOP functions with respect to the position of eight sensor nodes.

MATLAB's *fminunc* function was evaluated with two sets of initial conditions: one with nodes evenly spaced around the target, the other clustered in closely around the target. This was done to explore the solution's dependency on initial placement of the sensors. Each scenario was conducted five times to further illustrate the spread of

expected behavior. The results from range-only and bearing-only measurement scenarios are plotted in Figure 34 and Figure 35 respectively, with dots representing *fminunc* trial solutions and triangles representing the function's final solution.

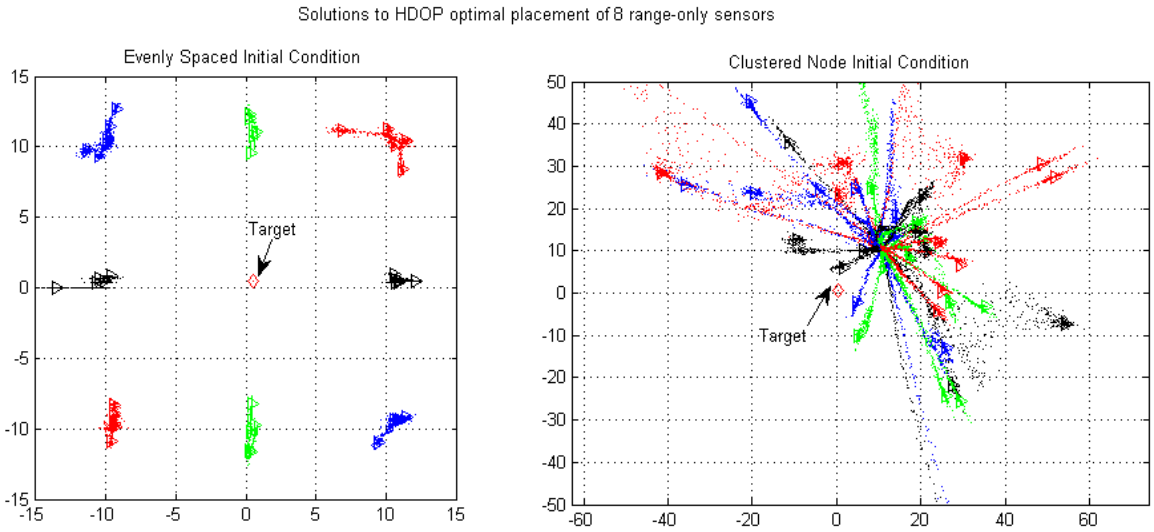

Figure 34. Range-only measurement: Minimizing solutions to HDOP-optimal placement of eight sensors with an evenly spaced initial condition (left) and clustered initial condition (right)

For range-only measurements, the evenly placed arrangement of the sensors is shown to be near-optimal. In the far left of Figure 34 it can be seen that the solutions held closely to their initial configuration with respect to the angle between the sensors. A slight extension in range from the target was observed, and this was expected for a range-only sensor. In the far right of Figure 34 we see the solvers attempt to enlarge the angle distribution from the initial clustered configuration while simultaneously maximizing range from the target. The result is a one-sided distribution of the nodes, as the solutions that attempt to expand to the opposite side of the target are likely discarded due to their near-range unsuitability.

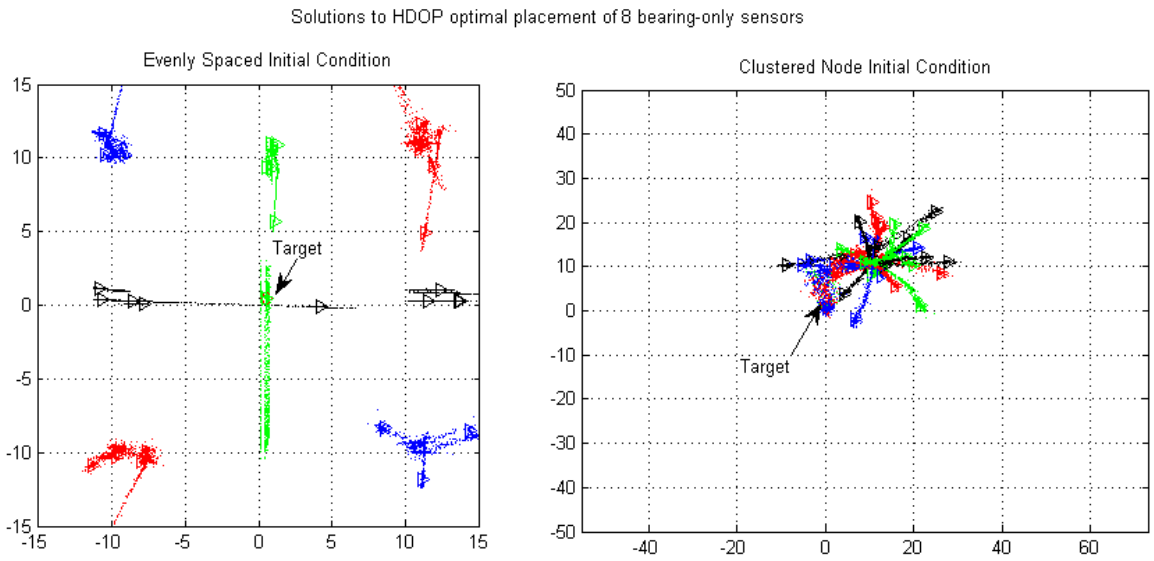

Figure 35. Bearing-only measurement: Minimizing solutions to HDOP-optimal placement of eight sensors with an evenly spaced initial condition (left) and clustered initial condition (right)

Considering the bearing-only measurement, we see evidence once again that the even distribution of sensors is nearly optimal in the far left of Figure 35. The intriguing difference between the two measurement scenarios is that for bearing-only there is consistently a node that attempts to get as close to the target as possible. This was somewhat expected due to the projected improvement of HDOP with decreasing range but also appears to be a tradeoff between the range and angle sensitivities of Equation (IV.10) as there was routinely only one node chosen to approach the target. The clustered distribution on the far right of Figure 35 shows a similar behavior, as the target is nearly covered up by the potential solution nodes.

This analysis highlights the angle separation dependences of both bearing-only and range-only measurement HDOP and also displayed some unexpected behavior in the bearing-only scenario. It is important to note that the solution provided by *fminunc* was largely dependent upon the initial conditions chosen. In addition, all of the discussions regarding HDOP assume the sensitivity of each sensor is range-independent. For example, increasing the distance to the target may improve HDOP with a range-only measurement, but in the case of ocean acoustics this also increases the transmission loss experienced by the signal, which may make its receipt by the sensor less probable.

B. RAPIDLY-EXPLORING RANDOM TREE DESCRIPTION

Leading up to this point we have discussed important elements of the thesis: energy balance, environmental parameters, and estimation filter characteristics. Each of them can now be integrated together under the realm of path-planning. In his ground breaking paper, LaValle [22] created the concept of a rapidly-exploring random tree (RRT), which sought to overcome some of the limitations of existing path-planning methods. Specifically focusing on a comparison to the probabilistic roadmap (PRM) approach, [22] points out that PRM algorithms attempt random configurations that must be connected to one another through the use of local planners. In the case where making these connections amounts to a nonlinear control problem, the undesirable increased computational complexity motivated the search for another method. The RRT algorithm was created to share many of the same advantages as the PRM method, such as minimal heuristics and limited arbitrary parameters. These features lead to repeatability and consistency [22]. However, the RRT relaxes the requirement to make a connection between adjacent states, making the RRT suitable for the nonholonomic and kinodynamic systems that often arise in robotics and reducing the computations significantly [22].

1. Basic RRT Algorithm Description

The RRT path planning method as described in [22] begins with a metric space X that can encompass any number of elements, to include variables like the orientation of a robot or its velocity and acceleration. In the general sense, X is not required to span a physical space and does not need to exist in only two or three visualizable dimensions. If a fixed obstacle region exists $X_{obs} \subset X$ that successful paths cannot pass through, it can be easily accommodated by the path planning approach. Once again, the concept of an obstacle is not limited to a physical object blocking a path and extends to any constraint the user specifies on X. Obstacles are also not explicitly defined in advance to the algorithm; the RRT only uses X_{obs} to check for collision when adding new elements to the tree.

The tree generated by the RRT is made up of vertices (points where connections are made) and edges (lines connecting adjacent vertices). All of the vertices added to the

tree will exist in X_{free}, which in a probabilistic sense is the complement of X_{obs}. Edges are created by extension from an existing vertex in the tree towards a target location. This extension is conducted via integration of a state transition equation $\dot{x} = f(x,u)$ that advances the state *x* with respect to a control input *u*. In many cases, including ours, Euler integration is utilized whereby $x_{new} \approx x + f(x,u)\Delta t$. Upon integration, if $x_{new} \subset X_{free}$ then $x_{new} \not\subset X_{obs}$ and therefore x_{new} can be added to the tree *T*.

To illustrate the mechanics of an RRT, we utilize the original psuedocode seen in Figure 36 from [22].

```
GENERATE_RRT(x_init, K, Δt)
1   T.init(x_init);
2   for k = 1 to K do
3       x_rand ← RANDOM_STATE();
4       x_near ← NEAREST_NEIGHBOR(x_rand, T);
5       u ← SELECT_INPUT(x_rand, x_near);
6       x_new ← NEW_STATE(x_near, u, Δt);
7       T.add_vertex(x_new);
8       T.add_edge(x_near, x_new, u);
9   Return T
```

Figure 36. Rapidly-exploring random tree pseudocode, from [22]

Consider the simple case of a bounded XY Euclidian geometric plane with obstacles. The state space being explored is the Cartesian position of the vehicle, [*X*, *Y*]. After the user defines a start and goal state, the tree is initialized in Step 1. At each of the iterations of Step 2, the RRT algorithm randomly selects a point in the plane x_{rand} in Step 3. It then finds the closest existing vertex in the tree x_{near} in Step 4 (at first iteration this will simply be the start point). Step 5 then extends an edge in the direction of that random point. The edge size itself is dictated by the control effort, *u*, but can be of a fixed size set by the user in this basic scenario. The algorithm creates the new state x_{new} in Step 6 and also checks to see if it impacts an obstacle. If it does not impact an obstacle, the

algorithm moves on to Step 7 and x_{new} becomes the newest vertex in the tree. Step 8 creates the edge between x_{near} and $x_{new.}$ This process repeats until the goal state is achieved or the number of specified iterations is exceeded. If the goal state is achieved, a final path is traced out from the start point to the goal whose vertices do not intersect an obstacle. Figure 37 is an example of the completed product from a basic fixed-step RRT algorithm, showing the exploration of space and the resultant path through an obstacle field.

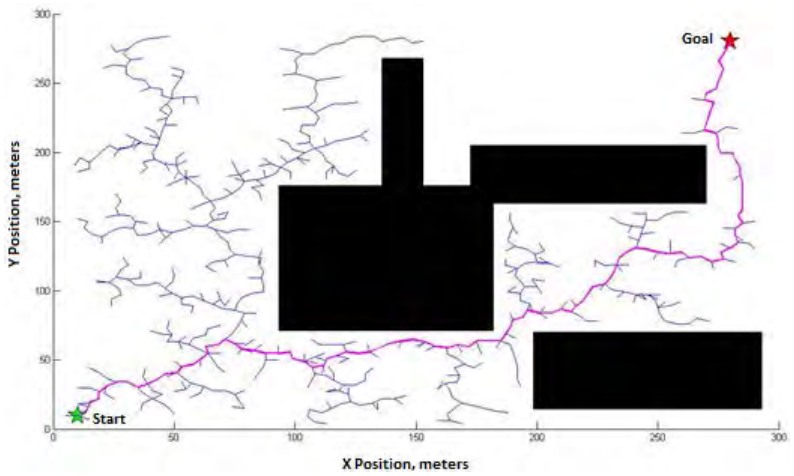

Figure 37. Path generated by a basic RRT algorithm through a fixed obstacle field

The behavior seen in Figure 37 is representative of some of the RRT's professed benefits: the bias towards unexplored space [22] is evident, none of the possible vertices intersect the obstacles, and a final path through the obstacle field has been found at the simple algorithm's conclusion.

From a theoretical standpoint, the RRT has proven probabilistic completeness [22]; however there are no guarantees on its optimality [26]. This is because the RRT does not take the "quality of the solution into account" by examining the cost of control efforts, time, and other metrics [21]. As a result of this, [21] proves that the RRT converges to a suboptimal solution the majority of the time and introduces the concept of RRT*, which achieves infinite-time asymptotic optimality. In simple terms, the RRT* does this by applying a cost to traversing each edge in the tree, while keeping a running

tally of the cumulative cost of using each path in the tree. The asymptotic optimality of the final path is guaranteed through the use of a rewiring step that looks back through the tree and reassigns vertices in the tree that grant a lower cumulative cost than their current configuration.

The RRT* concept suggests an excellent solution to the motion planning problem faced in the AquaQuad scenario. Asymptotic optimality provides an infinite-time guarantee on minimal energy expenditure while using the environmental disturbances to our benefit. The vehicle for crafting this behavior is the RRT*'s cost function, which not only takes into account the energy balance but also the quality of the flock's ability to track the target at each vertex that is contained in an HDOP calculation. Obstacle avoidance is built in to the algorithm, which will prevent interference from adjacent AquaQuads, in addition to avoidance of common at-sea obstacles such as islands, buoys, shipping routes, etc. In addition, these obstacle areas can represent locations of poor HDOP; any area we do not wish the AquaQuads to enter. Finally, the simplicity of the RRT facilitates its real-time usage and feasibility of hardware implementation. For this reason, and using concepts from [21], [22] and [27], we created an RRT* algorithm specific to our task.

2. Base RRT* Algorithm

This section seeks to describe the big picture steps that our RRT* algorithm uses, while the actual MATLAB code can be seen in Appendix C. In this discussion, we again refer to the easy to visualize case where the states being explored are within the XY plane. This plane is discretized into a number of individual cells, each of which contains information relevant to that specific location, including but not limited to the presence of obstacles and the predicted ocean current vector. The map is then initialized with a start and goal position. A representative example of the initial configuration is shown in Figure 38. In the following discussion, the trees created are not dependent upon the ocean current, but this is the style of figure will be used throughout this thesis. The goal position can be defined by the solution of a higher level operations research task that plans an optimal sequence of waypoints for a flock of coordinated AquaQuads in a given area of

operation. It can also represent a rapidly sampled point of minimal HDOP. Also of note, in the following discussion we use the term "node" interchangeably with the previously-used term "vertex" and substitute "branch" for "edge."

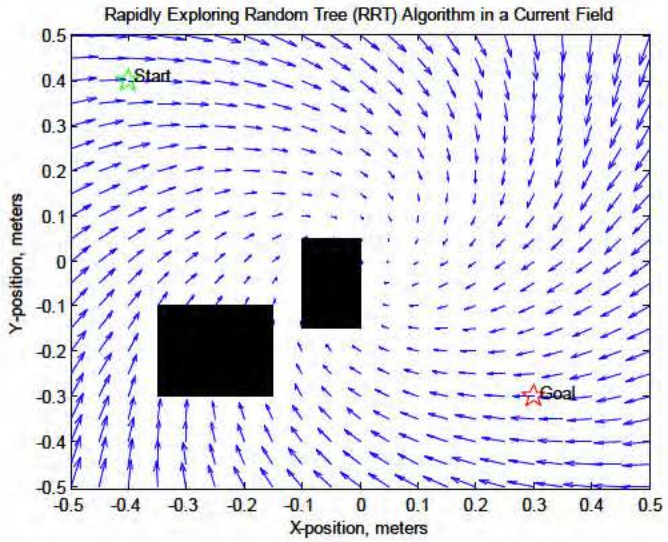

Figure 38. Initialization step of an RRT algorithm defining fixed obstacles, ocean current and the Start and Goal positions

a. Select Target with Defined Goal Probability

As the first step in our RRT*, a random "target" point x_{tgt} in the space is selected from a uniform probability distribution for analysis. This target is in fact pseudo-randomly selected, since the user will define that, for some small percentage of the time ("goal probability"), the target is forced to be the goal position x_{goal}. In this way the tree is biased towards the goal, which aids in faster convergence time at a small cost in space exploration. Figure 39 shows a partially completed tree with the pseudo-random target plotted.

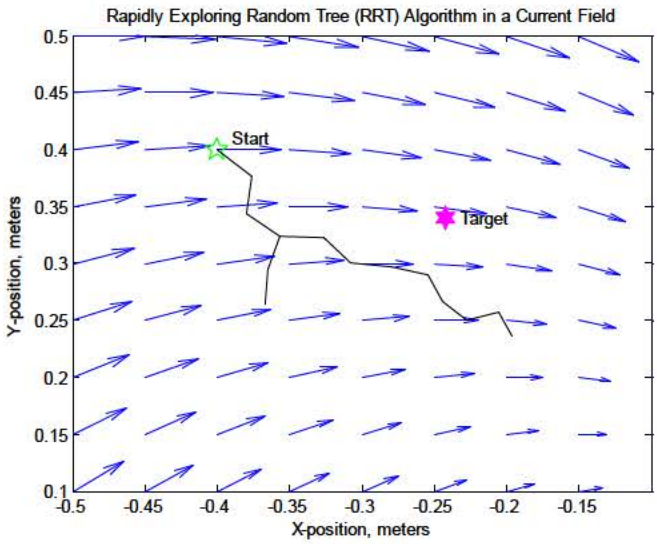

Figure 39. Early stages of RRT algorithm implementation showing the pseudo-random target selection step (Note: presented path not a function of ocean current)

b. Find Set of Closest Nodes and Determine Minimum Cost Node

Once the target is selected, the algorithm must select an existing node in the tree to extend from. In a basic RRT, it will look at the Cartesian distance between the target and all of the current nodes in the tree and select the closest one. In our RRT*, we select the closest "N" nodes and evaluate each to determine which one will result in the minimum cost, the function for which will be detailed in the next section of this thesis; for now it is sufficient to say that the best segment corresponds to longer drifting time along the given ocean currents. The value for N is defined by the user. Raising the number of nodes evaluated increases the likelihood of finding an optimal node but also increases computation time. The most cost-effective existing node becomes the parent node.

c. Extend Branch in Tree with Obstacle Detection

With the target and the parent node selected, the algorithm extends a branch from the parent node in the direction of the target. In a basic RRT, it will be of a fixed step size (i.e., – 1 meter). In our RRT*, we conduct simple Euler integration of the point-mass

kinematics of the vehicle over the time step we specify for the process. Limits can easily be defined for the control authority available to achieve this extension.

After integration, the position of the point-mass becomes a new candidate node x_{new}. We use the term "candidate" node to highlight that the node may not be added to the tree at this stage. Next, the location of x_{new} is checked to see if it falls within the limits of a defined obstacle in the space X_{obs}. If so, or if the control u required to achieve that point falls outside of the vehicle's available limits, it is thrown out, and the RRT* starts over with a new pseudo-random target, x_{tgt}. If x_{new} falls within the obstacle-free region X_{free}, it is added to the tree T as the newest node and a branch is connected between it and the parent node, as seen in Figure 40.

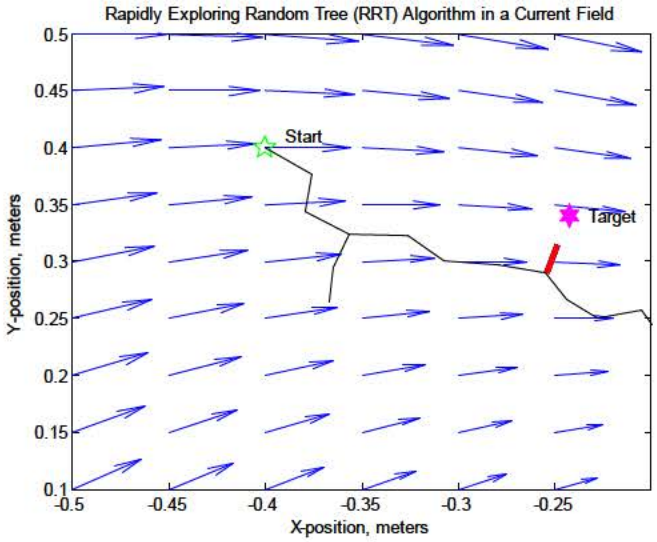

Figure 40. RRT algorithm showing obstacle free extension in red to pseudo-randomly selected target state (Note: presented path not a function of ocean current)

Stored within the tree is a node number that uniquely identifies the newest node, the number identifying its parent node, the cost of traversing the branch, and the cumulative cost associated with the path that goes from the Start point to this newest node.

d. Rewire Tree

The rewiring step occurs next, which provides the infinite time optimality guarantee. In our finite time application, this serves to improve the performance of our ultimate path. With the newest node now added to the tree, the rewire step selects existing nearby nodes in the tree that fall within a user-defined radius. If the cumulative cost of traveling to these nearby nodes could be reduced by reassigning the newest node as their *parent* node, it is updated as such and the old branch is discarded in favor of the new one. This process is very succinctly shown in Figure 41 and quoted from [28].

> when a new vertex is added to the tree, it is checked whether vertices that are already in the tree can be reached at a lower cost through this new vertex. This is also checked for a number of *k* nearest neighbors. In (Figure 41a) the tree is plotted again with the new vertex that has just been added and its *k* nearest neighbors. For these three vertices the costs of connection to the new vertex are given. It can be seen that vertex 6 is currently reached at a cost of 15, while through the new vertex this vertex can be reached at a cost of 12 (through vertices 0-1-5-9-6). Since this is lower than the current cost, the current edge toward vertex 6 is deleted and a new edge is added. The result is the tree of (Figure 41b). When a new edge is added to the tree a consequence is that the cost to reach the vertex that is rewired should be updated as well as the costs of all the child vertices of this vertex. [28]

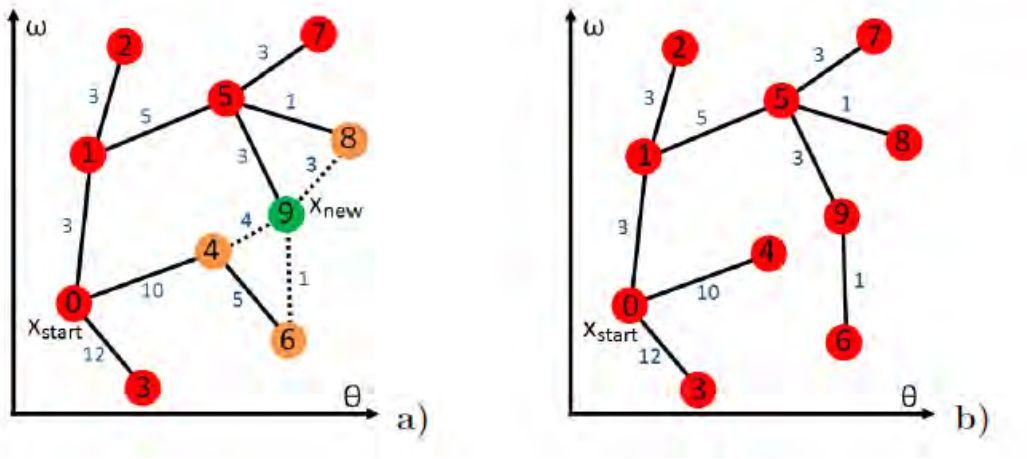

Figure 41. Visual display of RRT* rewire step, from [28]

e. Complete the Path and Reaching Goal or after "N" Iterations

After x_{new} is added to the tree, the algorithm checks if this node achieves the goal state. Typically, success is defined within a certain tolerance (i.e., -100cm), which can be adjusted based on the resolution required. If the goal is not met, and the maximum number of iterations is not exceeded, the cycle begins anew. When the goal is met, there are two choices. First, the algorithm can be terminated and the path from the start to the goal state is defined. Otherwise, the algorithm can continue searching for more cost-effective paths until directed to terminate. Since each node in the tree has its associated parent node stored alongside it, tracing the final path shown in Figure 42 is conducted by starting at the node that achieved the goal and working backwards.

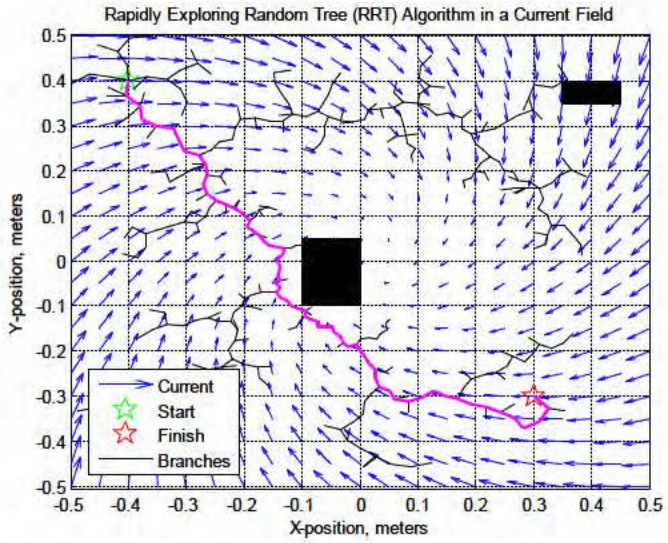

Figure 42. Final RRT path through a fixed obstacle field (Note: presented path not a function of ocean current)

C. DEAD RECKONING RRT* ALGORITHM

The preceding discussion touches upon many of the concepts that exist in the new DR-RRT* algorithm. However, our scenario required a specific variant that needed to take into account the limited control and energy available to the AquaQuads. Specifically, in order to use the oceanic currents to the maximum extent possible, we allow for extended drifting periods where we predict the future state of the vehicle given the

assumed ocean current vector at the node in the tree being analyzed. This is a basic "dead reckoning" (DR) step and like most DR solutions is a rough estimate whose accuracy is proportional to that of the ocean current map. That being said, the DR solution is integral to our algorithm, as the drift paths that are created can be linked via flight with "hop" steps. Figure 43 illustrates the difference between the DR-RRT* algorithm and a traditional RRT*.

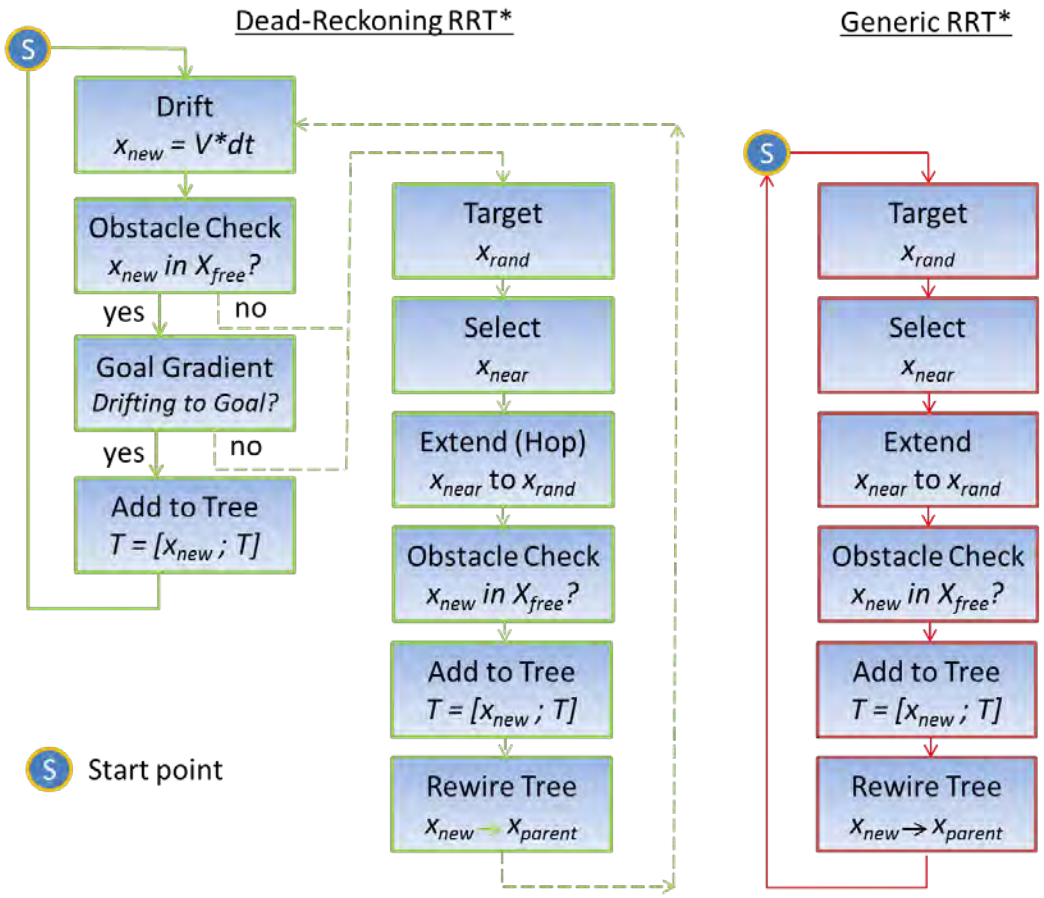

Figure 43. Comparison of the DR-RRT* algorithm utilized in this thesis with a generic RRT* algorithm.

The power behind the DR-RRT* algorithm is that it allows for periods where no control input occurs and hence minimal energy is expended. The bulk of the RRT* process is dormant during these phases, improving computational efficiency. Despite this dormancy, the nodes of the tree created while drifting are still used within the RRT*

framework and are essential to it. They represent obstacle-free paths with a defined cost associated with traveling each branch and provide candidate locations to both hop and rewire from once the RRT* is activated.

1. DR-RRT* Overview

As an introduction to our illustration of the DR-RRT*, Figure 44 displays the path created during the first DR phase of the algorithm. Shortly after this figure was generated, the ocean current begins to pull the predicted location of the AquaQuad into the center of the vortex, away from the red star denoting the goal/finish.

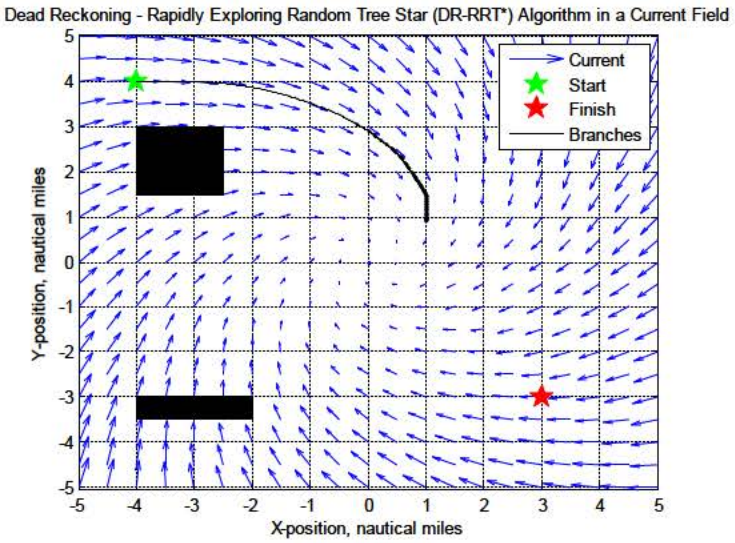

Figure 44. Drifting phase of the DR-RRT* algorithm with fixed obstacles, just prior to flight.

If it were possible to drift directly to the goal state, this would often be the best possible path. In lieu of that unlikely scenario, we implement a rule that verifies a negative gradient in range to the goal, or when the tree intersects an obstacle. Once the negative gradient is verified for an extended period of time, it activates the primary functions of the RRT* contained in the hopping phase.

When conducting the hop phase, the AquaQuad is allowed limited flight time. Flight is heavily penalized, commensurate with the increased energy expenditure. The

direction of that flight is pseudo-randomly selected in the standard RRT* manner, it originates from the most cost-efficient nearby parent node, and the final position of the AquaQuad is then tested for obstacle presence. The random nature of the flight steps is important, because it allows us to explore the configuration space and evaluate the cost of different paths. The final result of a hopping step is shown in Figure 45. A completed hop step is visualized as a green arc connected by red circles that display takeoff and landing locales.

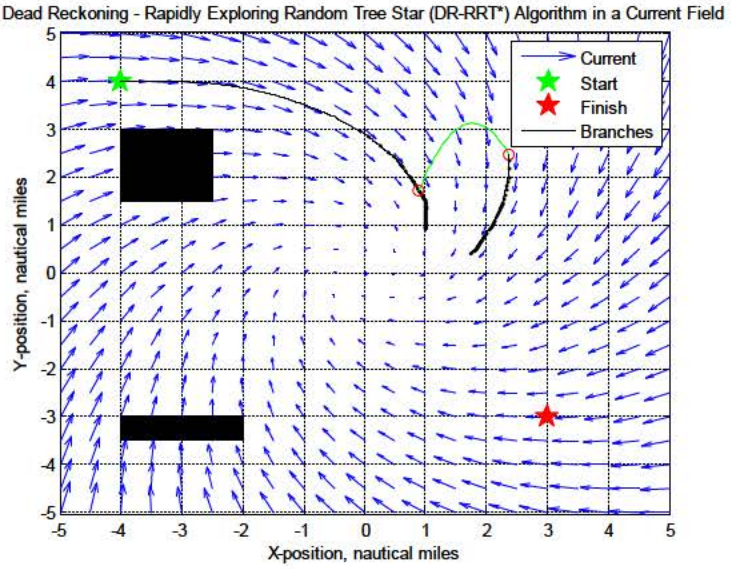

Figure 45. Hopping phase of the DR-RRT* algorithm with fixed obstacles. Flight path shown as green arc from one drifting path to a new one.

Rewiring also occurs in the hopping phase, whereby drifting paths in adjacent branches can be joined by flight to the newest node if it results in a lower cumulative cost. In order to preserve the continuity of final paths in our version, rewiring is only allowed on nodes that originate from or terminate in a hop step. This ensures that control authority exists for the AquaQuad on the new path. The result of one simulation of the DR-RRT* algorithm, which terminates once the Goal point is achieved, is shown in Figure 46. In this case, the obstacle is located near the center of the ocean current vortex, so that many of the black drift paths are unsuitable without flight. The final path created

by the algorithm is shown in magenta and requires four periods of flight whose combined length is less than that required for direct flight.

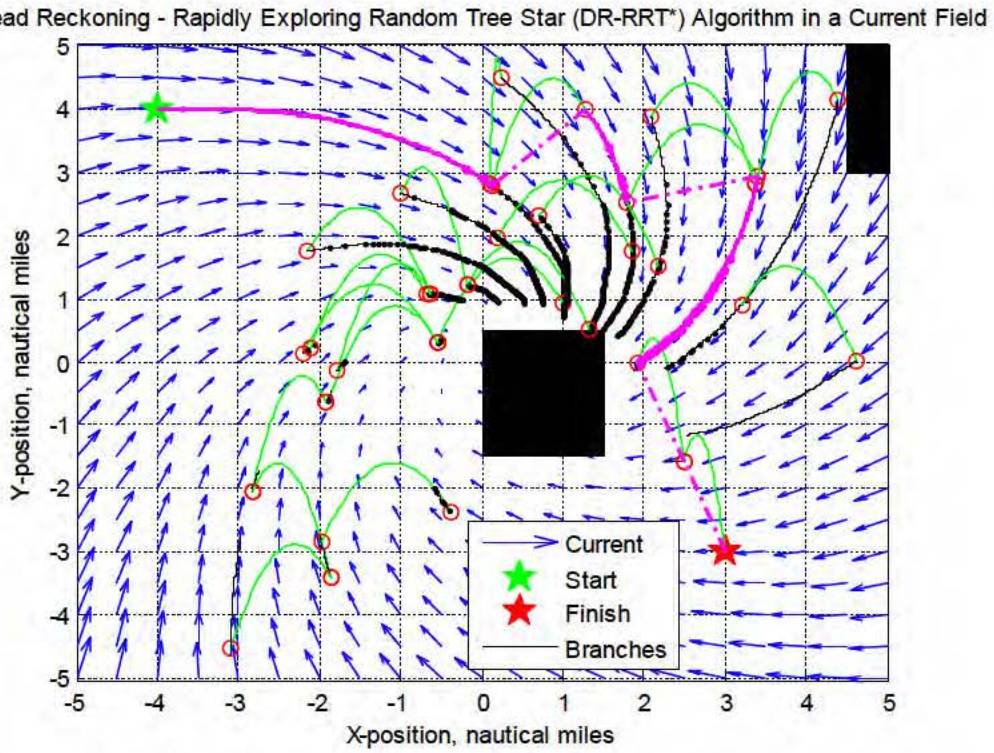

Figure 46. DR-RRT* algorithm with fixed obstacles. Final path is shown in magenta. Evaluated flight paths are represented as green arcs. Evaluated drift paths are represented as back lines.

2. DR-RRT* Objective Function

One of the useful features of the RRT* is that the objective or "cost" function, can be utilized to shape the paths that are generated to the goal. The use of objective functions is a common way of combining multiple criteria to be optimized into a weighted and linearly combined sum. For our purposes, we desire paths that are energy-efficient, so we began with the energy balance shown in Equation (IV.14) that is based upon the discussion in Chapter II.C.

$$FlightEnergy = 200W\left(\frac{FlightDist, nm}{20kts}\right)$$
$$SolarEnergy = 400Wh\left(\frac{1}{24hrs}\right)(DriftTime, hrs) \quad \text{(IV.14)}$$

We also desire paths that have desirable geometry for the tracking mission (i.e., minimal HDOP) and of course those that reach the goal point. The energy terms are a function of time spent in a drifting or flying state, however the HDOP and distance to goal terms are a function of position. These computations are conducted in real-time depending upon the location of the node in the tree being analyzed, however Figure 47 and Figure 48 give a three-dimensional view of how distance to the goal and bearing-only measurement HDOP change as the position of a generic "QuadRotor #4" changes.

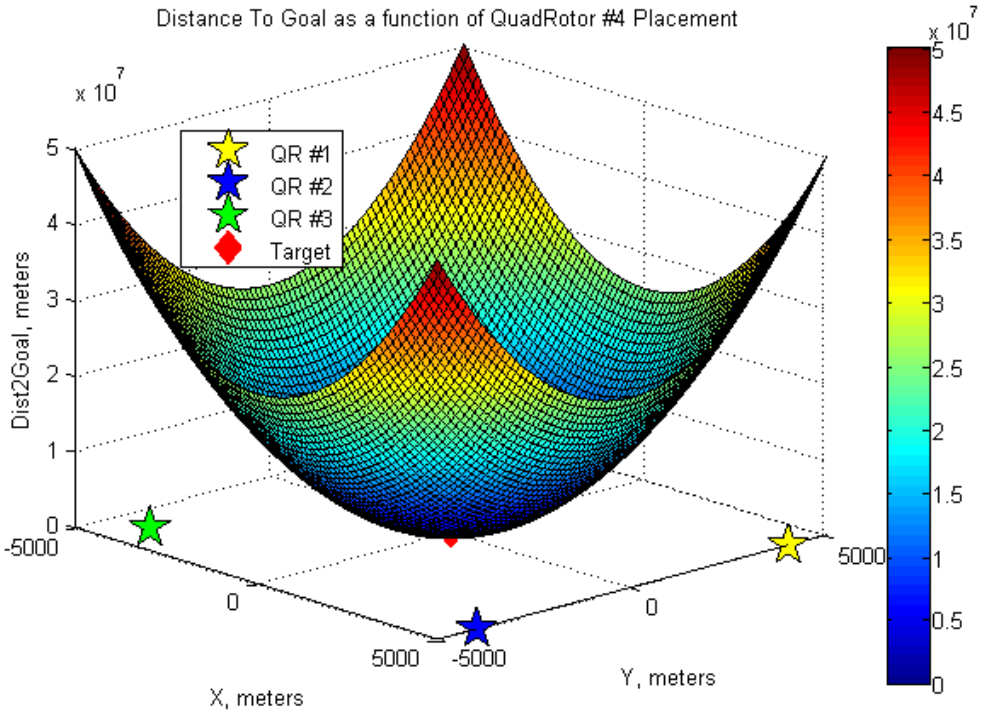

Figure 47. Value of the objective function element "Dist2Goal" with adjusted Quadrotor position in the configuration space

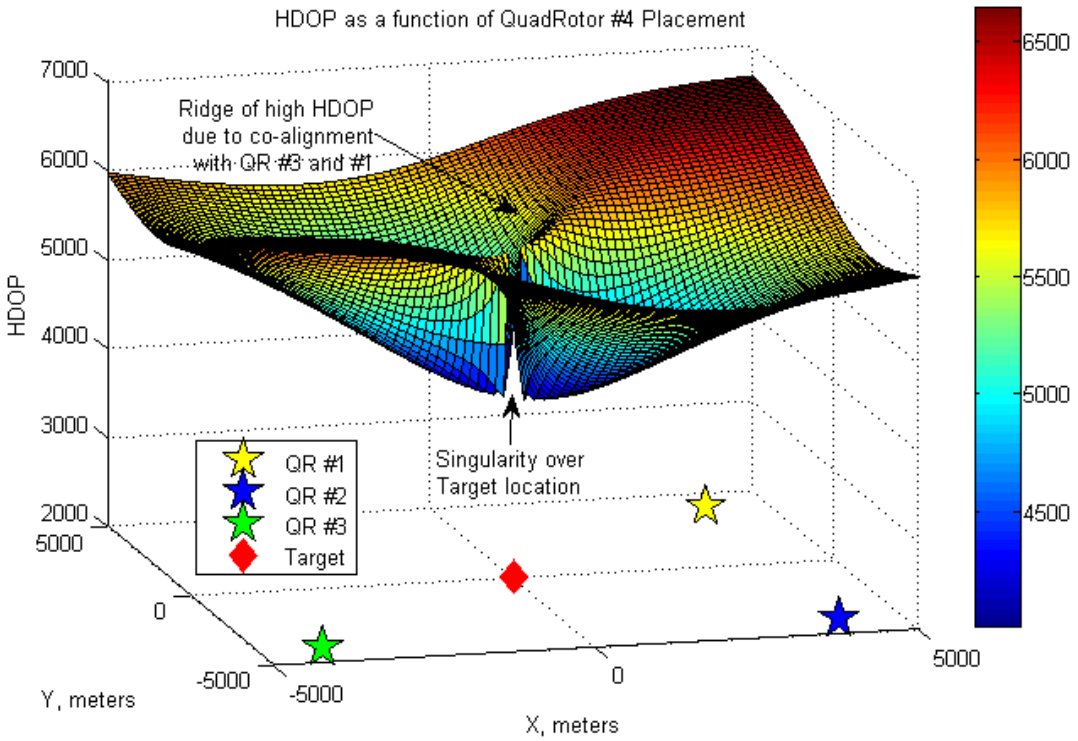

Figure 48. Value of the objective function element "HDOP" for Bearing-only measurements with adjusted Quadrotor position in the configuration space

The HDOP values in Figure 48 are of particular interest, as it can be seen that for bearing-only measurements (see Figure 32 for the case of TDOA measurements), a singularity exists when QuadRotor #4 is directly on top of the target. In addition, a ridge of high HDOP is present that spans from the location of QuadRotor #1 to QuadRotor #3. This is a visual indication of the degradation in HDOP due to co-alignment between sensors, see discussion in Chapter IV.A.1.

With the terms for HDOP and distance to the goal (*Dist2Goal*) now included, we normalized all components and scaled each of the terms with a gain K_i as seen in Equation (IV.15).

$$J = \int_{t_o}^{t_o+\Delta t} K_1\left(\frac{HDOP}{HDOP_{MAX}}\right) + K_2\left(\frac{Dist2Goal}{Dist_{MAX}}\right)\ldots$$
$$+K_3\left(\frac{FlightEnergy}{FlightEnergy_{MAX}}\right) - K_4\left(\frac{SolarEnergy}{SolarEnergy_{MAX}}\right)dt \qquad \text{(IV.15)}$$
$$where \ \sum_{i=1}^{4} K_i = 1$$

Note the negative sign associated with the solar energy term K_4 of Equation (IV.15). This is to account for the energy *gain* associated with drifting. Longer drifting periods minimize the cost of travel and create a desirable bias towards the selection of these paths.

One of the common issues with the use of objective functions is that they are very sensitive to weighting and this can have effects upon their optimality [29]. Therefore, determination of the magnitude of each K_i term was conducted empirically. Some interesting results were generated during these trials. Specifically, if the K_4 term was too large, Equation (IV.15) became a decreasing function with distance traveled. This had undesirable side effects during the Rewire step, where loops were made in the final path that did not reach the goal. Figure 49 illustrates this behavior with generic values for the cumulative cost at each point of consideration.

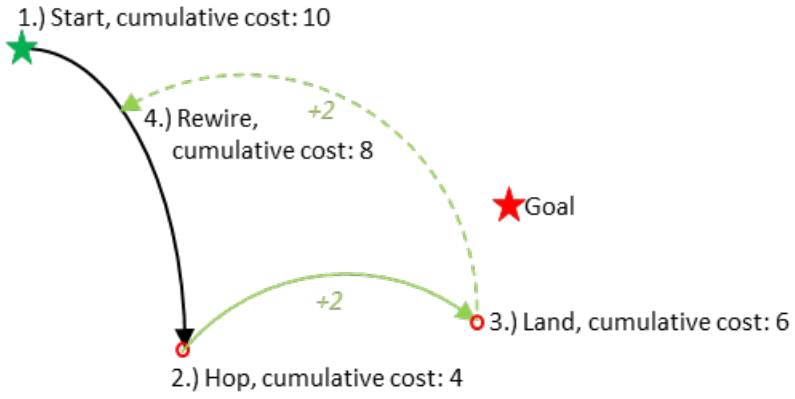

Figure 49. Improper rewire behavior exhibited with excessive weighting on solar energy term of cost function in DR-RRT* algorithm

Drifting from Point 1 in Figure 49, there is an accumulation of *negative* cost associated with absorbed solar energy that drops the cumulative cost of reaching Point 2 down to "4." At this point, the algorithm determines a Hop is necessary (due to increasing distance to the Goal point or obstacle impact), and flight is allowed to Point 3 at an additional cost of +2. At this point, as described in Section IV.B.2, the DR-RRT* looks to rewire with a nearby node in the tree. Point 4 is improperly selected for rewire since flight raises the cumulative cost of obtaining this point to only "8", which is less than that obtained by drifting from Point 1 to Point 4 at the onset. It can be seen that this creates an infinite loop from 2-3-4-2 that does not reach the goal.

Table 3 shows the final K_i terms selected for our DR-RRT* and the qualitative justifications for their selection. In addition, it should be stated that during drift paths the K_3 term is switched to zero as no flight occurs in these phases and once again that the cost function of Equation (IV.15) has a negative sign associated with K_4. The MATLAB implementation of the DR-RRT* code can be found in Appendix C.

	Value	**Associated Term**	**Justification**
K_4	0.005	Solar Energy In	Small value to maintain cost an increasing function with distance traveled
K_3	0.95	Flight Energy Out	Large value to heavily penalize flight
K_2	0.0225	Distance To Goal	Bias towards paths which grow towards the goal
K_1	0.0225	HDOP	Bias towards paths with minimal HDOP for improved tracking

Table 3. Gain terms applied in cost function of DR-RRT* algorithm with justifications for usage

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULATION AND TESTING

The algorithms created in this thesis were designed with the ultimate goal in mind that they will be fielded on the future AquaQuad. To that end, we seek to explore the real-time implementation of the UKF and DR-RRT* concepts in live testing. We also require some validation of the DR-RRT*'s professed energy-efficiency benefits for the AquaQuad. This chapter details those steps and their results.

All simulations were conducted utilizing MathWorks' MATLAB and SIMULINK software utilizing a Dell Optiplex 790 desktop computer with 4 GB RAM and an Intel Core i7-2600 processor operating at 3.40 GHz with a Windows 7 64-bit operating system. All testing was conducted at the Naval Postgraduate School's Center for Autonomous Vehicle Research in Monterey, CA.

A. DR-RRT* ALGORITHM ANALYSIS

The RRT*'s guarantees on optimality are tied to the concept of infinite time. In the finite-time world application, optimality is sought but not guaranteed. The RRT* can terminate once a solution is found, or it can continue to find paths that further minimize the objective function. This comes at the cost of computation time, whereas it may be preferable to find a rapid "good enough" solution instead. A simple question is then: How much is "good enough"?

1. Single-solution DR-RRT*

To explore the gains obtained by running the DR-RRT* beyond its initial solution point, we needed to first gather some statistics on the single-solution algorithm. To do this, we used a Monte Carlo approach where the data from 1,000 simulations was collected for analysis.

An initial obstacle set was randomly generated and then fixed in place for future simulations to ensure consistency. The goal probability (defined in Chapter IV.B.2) was set at 0.05, so the finish point was selected as the random target for flight approximately 5 percent of the time. The positions of three other AquaQuads were simulated at (2.5,

2.5), (2.5, -2.5) and (-2.5, -2.5) respectively as inputs to the HDOP calculation. At the algorithm's conclusion sequential flight segments were combined.

Four of the successful trees are displayed in Figure 50. Once again, the black lines represent unsuccessful drift paths, and the green arcs represent unsuccessful "hop" steps of limited flight. At the algorithms conclusion, sequential hop segments were joined into a single period of flight. The final successful path is shown in magenta. The four trees shown are useful to highlight some expected tendencies of the DR-RRT*.

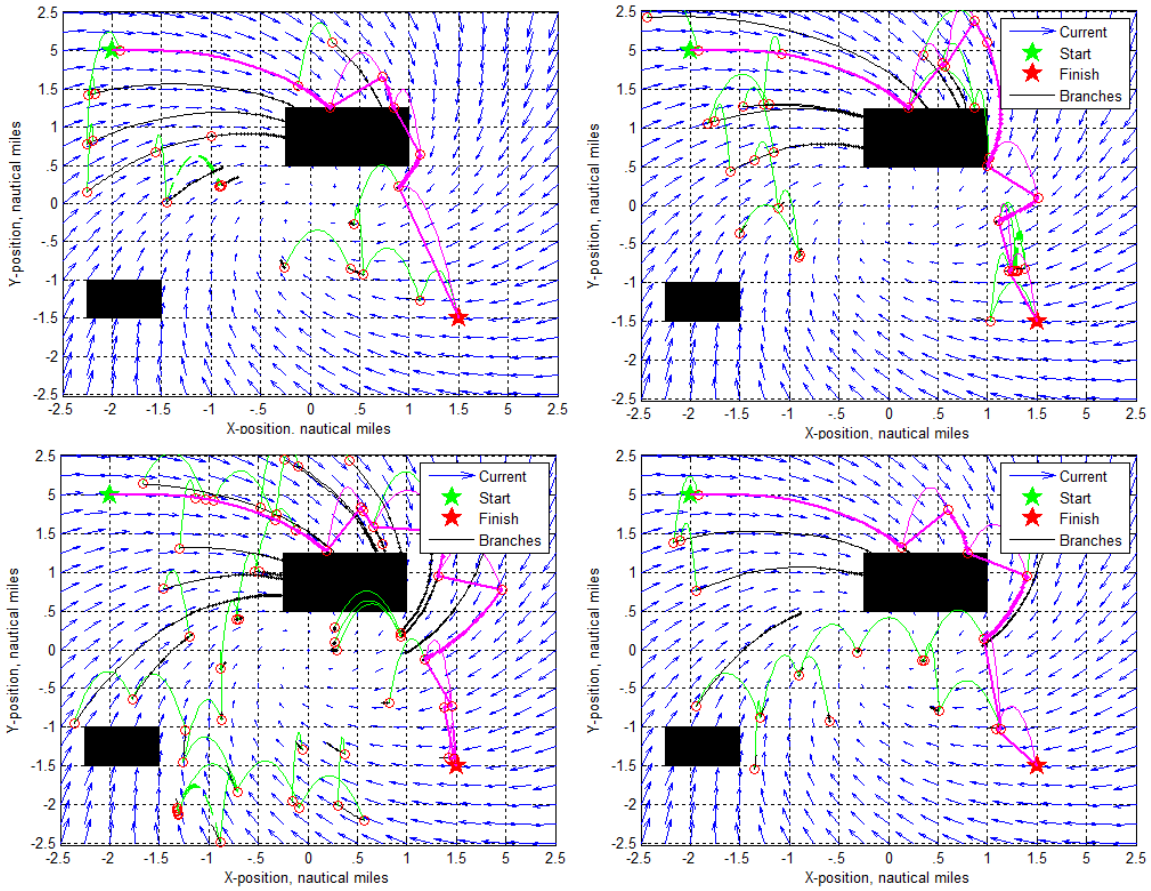

Figure 50. Successful paths (shown in magenta) obtained from a Monte Carlo simulation conducted using the DR-RRT* algorithm with a fixed obstacle field

Given the vortex ocean current flow, it was anticipated that successful paths would not be generated from the lower left-hand quadrant. In these regions the current

flow drives the AquaQuad away from the goal, which is undesirable from both a conceptual and mathematical standpoint, since these paths maximize the *Dist2Goal* term of the objective function and force hop steps. This behavior is evident in Figure 50, as all successful paths occur towards the right of the obstacle; a region where ocean current tends to drive the AquaQuad towards the finish. Another trend of interest is the tendency for the AquaQuad to often fly directly to the goal once the tree crosses the x-axis into the negative y region. All ocean current vectors drive away from the Goal in this location, so the algorithm aggressively hops and ultimately strikes the finish marker.

The primary data points of interest from the Monte Carlo simulation are the energy expended in the successful path and the time it took to run the algorithm to its conclusion. These values were stored at the completion of every tree that reached the Goal and are displayed in Figure 51.

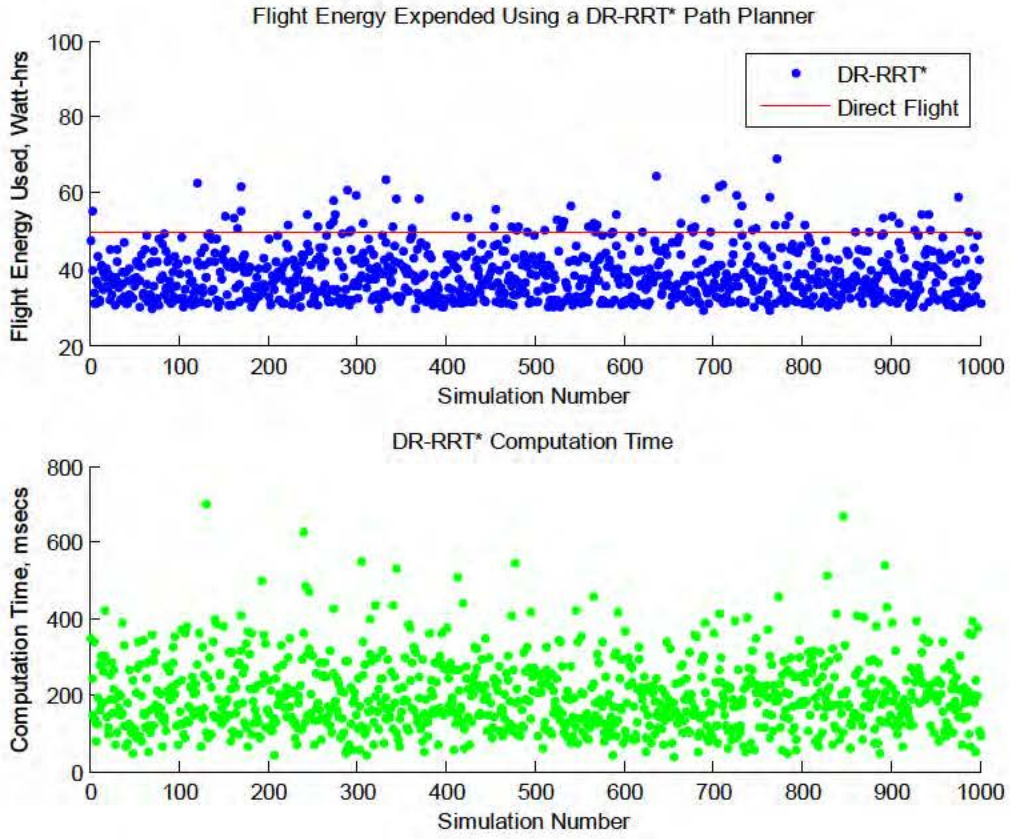

Figure 51. Energy expenditure and computation time of 1,000 runs of the DR-RRT* algorithm. Direct flight baseline is shown in red in the top figure and represents flying the straight-line distance between Start and Goal points

The red line at the top of Figure 51 represents the 49.5Wh of energy that would be expended in direct flight between the Start and Goal positions. Overall, the results of the simulation were very positive. The mean energy value of the sample set was 38.69Wh with a standard deviation of 6.81Wh, an improvement of approximately 22 percent. Computation time was short, with a mean value and standard deviation of 190ms and 90ms, respectively.

It can be seen in the top portion of Figure 51 that relatively infrequently the DR-RRT* found paths that utilized more energy than that consumed during direct flight. In application, the algorithm could be forced to run again. An alternative viewpoint is that since traversing the DR-RRT* paths take several hours of time, a staggered approach to

flight may be more desirable. During that time the AquaQuad will be contributing measurements to the tracking scenario and absorbing solar energy during daylight hours, and the path it is following will be obstacle-free. Conversely if the AquaQuad were to fly directly to the goal point its drifting status would be unregulated without further planning. This raises the risk of impacting an obstacle or drifting to areas of poor HDOP.

2. Multi-solution "Optimality-Seeking" DR-RRT*

With the statistics from the single-solution DR-RRT* determined, we then seek a qualitative and quantitative comparison to the result of a DR-RRT* simulation that is allowed to run far past its initial solution. In doing so, the algorithm will continue to find new paths to the goal while rewiring existing paths in order to make the final path closer to optimal. We use the term "optimality-seeking" for this evaluation, and we expect that the minimum-cost path from the optimality-seeking DR-RRT* will improve upon the mean value of the single-solution DR-RRT* energy consumption at the expense of computation time.

We again utilized a Monte Carlo approach to develop statistics, collecting data from 1000 simulations of the optimality-seeking DR-RRT*. Each simulation was allowed to run for 25,000 iterations and the successful paths to the goal were aggregated. The path with the minimum energy expenditure that reached the goal was selected, the flight energy expended on that path and computation time recorded, and a new simulation was conducted. Four of the successful trees are displayed in Figure 52.

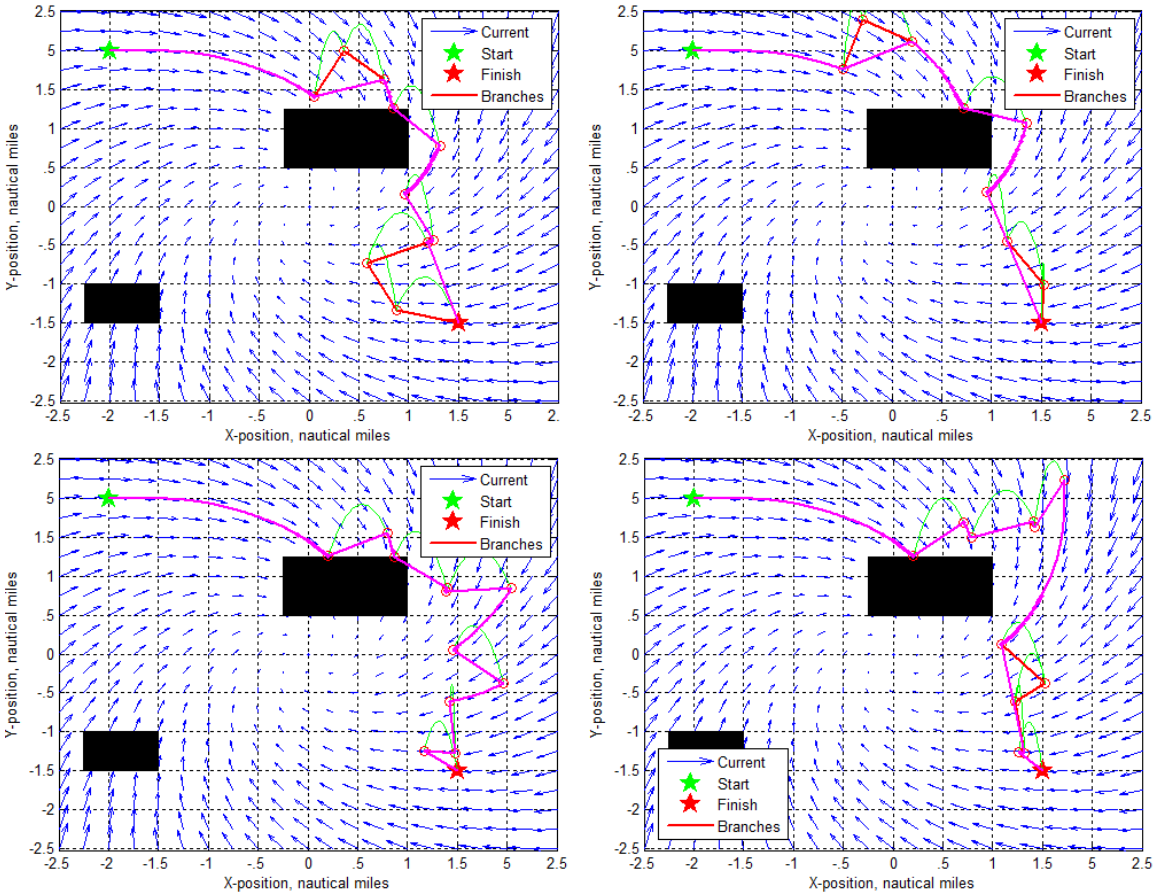

Figure 52. Successful paths (shown in magenta) obtained from a Monte Carlo simulation conducted using the optimality-seeking DR-RRT* algorithm with a fixed obstacle field. Candidate paths omitted.

Unlike Figure 50, Figure 52 does not have the candidate paths drawn, only the final solutions. Once again, we combine sequential hop segments in our final path into a single period of flight (this behavior is evident in the top left tree of Figure 52). Figure 52 does share many of the same characteristics as its single-solution counterpart, namely the bias in the path towards beneficial ocean current vectors. The true depiction of its proposed benefit comes from the underlying data that is seen in Figure 53.

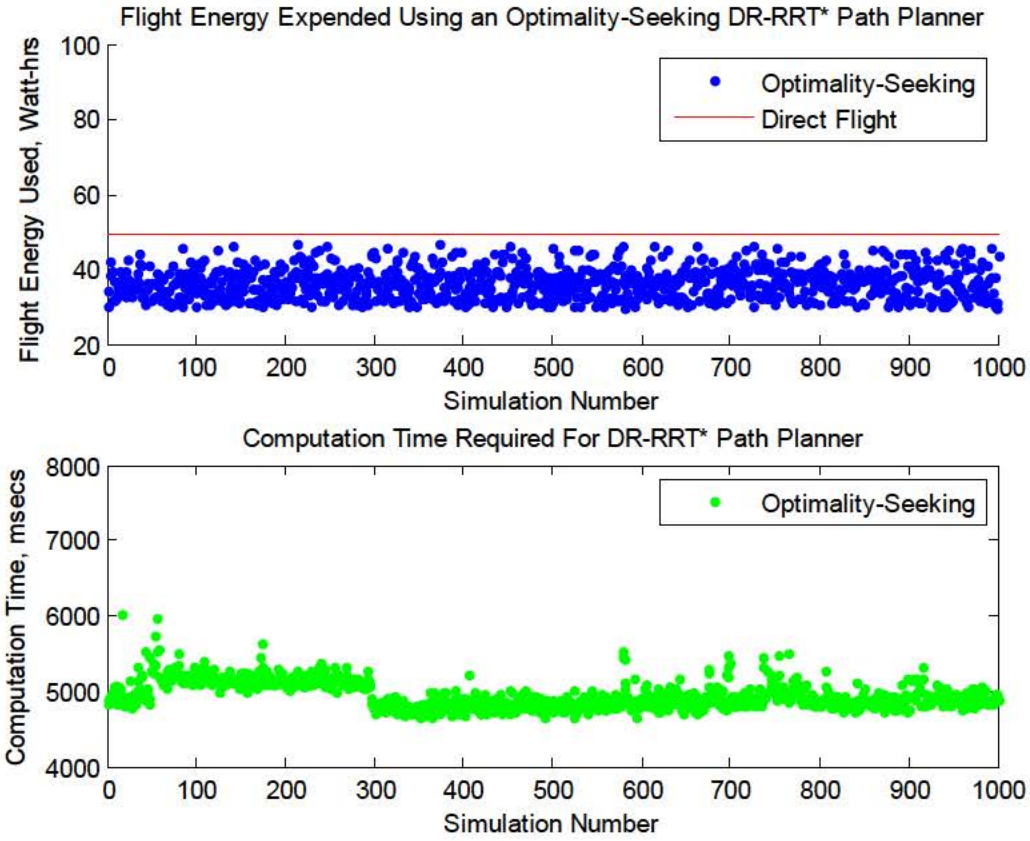

Figure 53. Energy expenditure and computation time of 1,000 runs of the optimality-seeking DR-RRT* algorithm. Direct flight baseline is shown in red in the top figure and represents flying the straight-line distance between Start and Goal points

Once again, the solid red line signifies the 49.5Wh of energy expended in direct flight. Of particular interest is the fact that none of the 1,000 simulations produced a path that used more energy than direct flight. The mean energy value of the optimality-seeking DR-RRT* sample set was 36.26Wh with a standard deviation of 3.93Wh. This is an improvement of 27 percent over direct flight but only six percent over the single-solution DR-RRT*. This energy efficiency came at a small price, as the mean computation time for the sample set was almost five seconds: 4,964msec with a standard deviation of 18.12msec, an increase in the mean of almost 2,500 percent over the single-solution DR-RRT*.

3. DR-RRT* Evaluation Summary

Table 4 provides a summary of the pertinent data points collected in the Monte Carlo analysis of the DR-RRT*. The benefit from running the algorithm beyond its initial solution point is irrefutable, but these gains must be weighed against the computational cost. The value of computation time is user-dependent and linked to the microprocessor the code is being implemented upon.

	Single-Solution DR-RRT*			**Optimality-Seeking DR-RRT***		
	Mean	Standard Deviation	Improvement over Direct Flight[†]	Mean	Standard Deviation	Improvement over Direct Flight[†]
Energy, Wh	38.69	6.81	22%	36.26	3.93	27%
Computation Time, msec	190	90	-	4964	18	-

[†]Direct flight utilizes 49.5Wh of energy flying directly to goal point

Table 4. Summary of Monte Carlo simulation results contrasting a single-solution DR-RRT* algorithm with that obtained by an optimality-seeking DR-RRT*. Percent improvement based upon a direct flight comparison.

B. TESTING ENVIRONMENT AND CONTROL OVERVIEW

This section discusses the environment, platforms, and control structure that we used to explore real-time implementation of our algorithms. Despite being evaluated under slightly different scenarios, these elements were common to the testing of both the UKF and DR-RRT* concepts and are therefore summarized here.

1. Environment: Center for Autonomous Vehicle Research

Testing was conducted in the lab space of the Center for Autonomous Vehicle Research (CAVR). This space is instrumented with a Vicon motion tracking system [30] that provides high resolution position (*x, y, z)* and orientation *(roll, pitch, yaw*) estimates of objects in the camera's cumulative frame of reference.

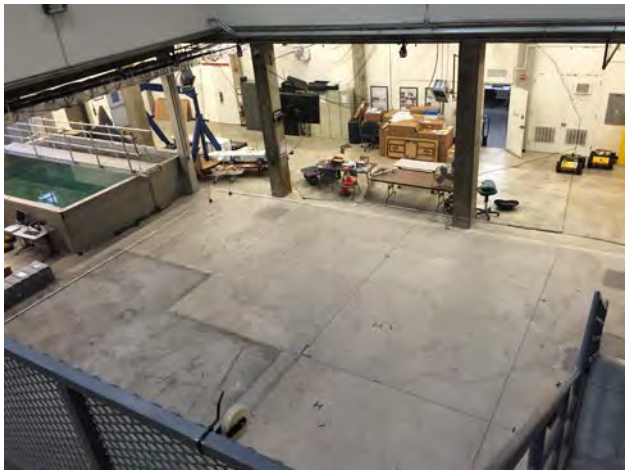

Figure 54. Vicon instrumented laboratory in the Naval Postgraduate School's Center for Autonomous Vehicle Research

To determine the pose of objects in the field, Vicon utilizes several optical cameras mounted at high points in the lab space. Each of these cameras (far left of Figure 55), has rings of LED strobe lights surrounding the lens that cycle at a high frequency. The light produced by the strobe is reflected off of markers affixed to objects in the field of view (center of Figure 55), filtered through the lens and captured by a light sensitive plate behind it [30]. This data is transmitted to a workstation PC (far right of Figure 55) for reconstruction.

Figure 55. Two of the CAVR lab's Vicon cameras (far left), reflectors affixed to a quadrotor shroud (center) and the Vicon Workstation (far right)

The object data produced by the Vicon Workstation is sent via an Ethernet router to the user's personal computer and can be ingested into SIMULINK via User Datagram Protocol (UDP). For our purposes, the Vicon system facilitates testing of the UKF and DR-RRT* algorithms in real-time through pseudo-measurements and control inputs that are derived from the positions of all players in the field of view.

2. Platforms: Parrot AR.Drones

With the AquaQuads under construction, we required a proven commercial substitute for testing, and Parrot's AR.Drone platform is just that. The internal sensors on board of the AR.Drone record and wirelessly transmit body velocity (u, v, w), orientation (*roll*, *pitch*, *yaw*), altitude (z), and remaining battery life. Of perhaps larger importance is their ability to also accept commands wirelessly, allowing them to be controlled with a high degree of authority.

Figure 56. Parrot AR.Drone quadrotor utilized for testing purposes

3. Control Structure

The overarching control scheme for the AR.Drones is depicted in Figure 57, which highlights the flow of communications. The position and orientation of each quadrotor was provided from the Vicon system at a frequency of 100Hz. This was used as an input to the master laptop, which took the current state of the quadrotors and produced control inputs to them at a frequency of 1Hz.

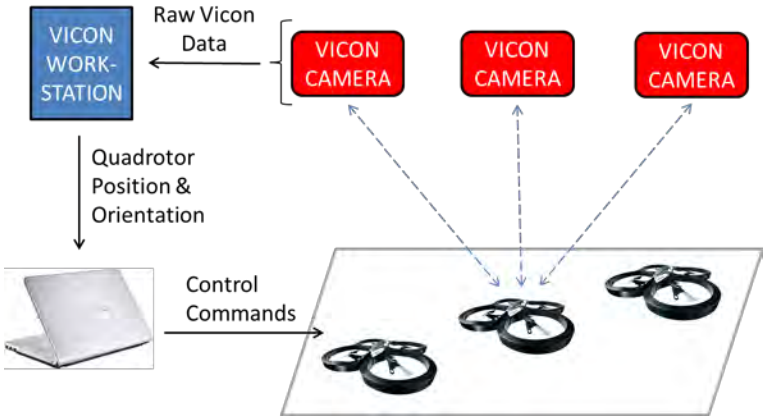

Figure 57. Communications overview for testing in the CAVR laboratory

Control commands for the quadrotors were produced utilizing proportional-integral (PI) and proportional-integral-derivative (PID) controllers within the SIMULINK

environment that sought to minimize the error between the state of the quadrotor and a reference signal.

Vicon provides the position of the quadrotor within the camera's frame of reference. Commands provided to the AR.Drones needed to be translated into their fixed body frame of reference whose origin is located at the center of the quadrotor itself. The interested reader should consult Chapter 2 of either [1] or [18] for a full treatment of reference frames and translation between them. For our purposes, translation between our two right-handed coordinated frames is conducted using a direction cosine matrix of the form

$$\begin{bmatrix} {}^{b}x \\ {}^{b}y \\ {}^{b}z \end{bmatrix} = {}^{b}_{c}R \begin{bmatrix} {}^{c}x \\ {}^{c}y \\ {}^{c}z \end{bmatrix} \tag{V.1}$$

where *c* represents the camera frame, *b* represents the body frame and

$${}^{b}_{c}R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ & \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{V.2}$$

with ϕ, θ and ψ representing roll, pitch, and yaw respectively.

The block diagram representing the SIMULINK control system is shown in Figure 58. The AR.Drone accepts heading rate $\dot{\psi}$, roll angle ϕ, pitch angle θ, and vertical velocity V_Z commands. The PI and PID controllers take heading and position references and subtracts the current state of the quadrotor to produce an error term. This signal is then sent through raw (no alteration of error term) proportional, derivative, or integrated channels, each multiplied by empirically-determined gains to produce the desired commands. Of note, the velocity of the quadrotors utilized in the PID controller was determined by taking the derivative of the Vicon position signal and then utilizing a low-pass filter to remove the higher frequency oscillations, thereby smoothing it. The primary style of implementation for all controllers is feed-back, complemented with a feed-forward controller in the case of roll and pitch commands.

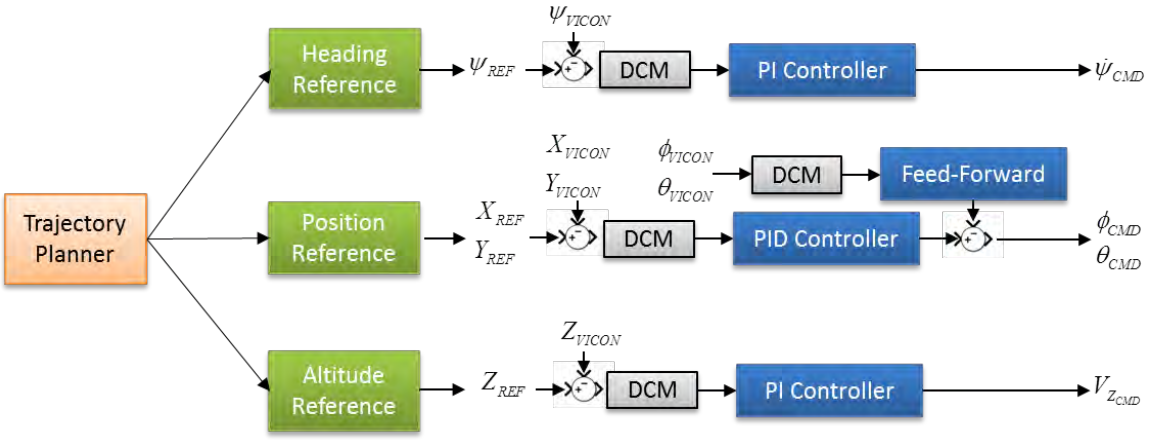

Figure 58. Block diagram of SIMULINK control implementation for AR.Drone quadrotors

The Trajectory Planner in Figure 58 is left as a generic entity that was modified depending upon the test being conducted and will be discussed in the following sections.

C. UNSCENTED KALMAN FILTER TEST

The first test conducted in the CAVR lab was designed to evaluate the UKF-based tracking algorithm on the quadrotors. We present the envisioned at-sea scenario that the test is founded upon, provide some relevant implementation details, and then discuss its results.

1. Unscented Kalman Filter Scenario

In the UKF test scenario, AquaQuads with TDOA measurement capabilities are surrounding a target that is undergoing random motion. This target behavior differentiates it from the linear motion simulated in the EKF/UKF comparison of Chapter III.E. For this purpose, we placed each of three quadrotors in a Y-shaped configuration around the object to be tracked in the center. The basic setup is shown in Figure 59.

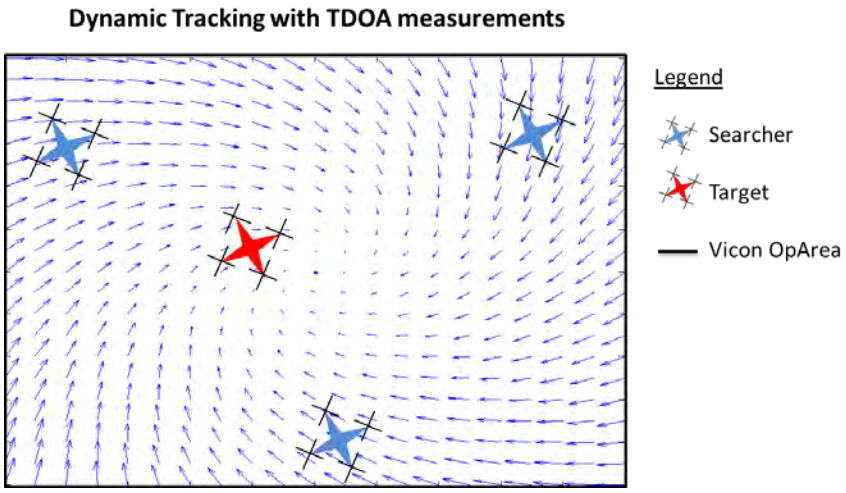

Figure 59. General scenario overview for the UKF tracking test showing the position of sensors and the target (ocean current vectors not utilized in CAVR lab)

In this test the ocean current vectors were not utilized, however each of the searcher quadrotors was commanded to hold its position at an altitude of two meters in order to simulate a floating AquaQuad. The quadrotors were also commanded to change their heading to point in the direction of the estimated position. When the heading of each quadrotor and the target location aligned, this provided a visual representation of the filter's convergence. These commands make up the function of the "Trajectory Planner" block of Figure 58 for the UKF scenario.

The UKF equations from Figure 23 were modified for TDOA measurements and incorporated within an embedded MATLAB function block for SIMULINK implementation. This can be seen in Appendix B. In order to obtain TDOA pseudo-measurements that would approximate those received by an actual sensor, we took advantage of the Vicon position data available to the lab. Specifically, at every call to the UKF the true range between the target and each quadrotor was determined. These ranges were divided through by a common value for the speed of sound in the ocean (1500m/s) to produce the time in seconds that it would take a signal to arrive at the AquaQuad. These three separate reception times were then subtracted from one another to produce a time-difference of arrival value τ_{ij}, as discussed in Equation (II.7). Lastly, band-limited white noise was added to the measurements with a power of 10^{-8}, incorporating error into

the measurement that the UKF would need to filter out in the estimation process. The SIMULINK implementation is shown in Figure 60.

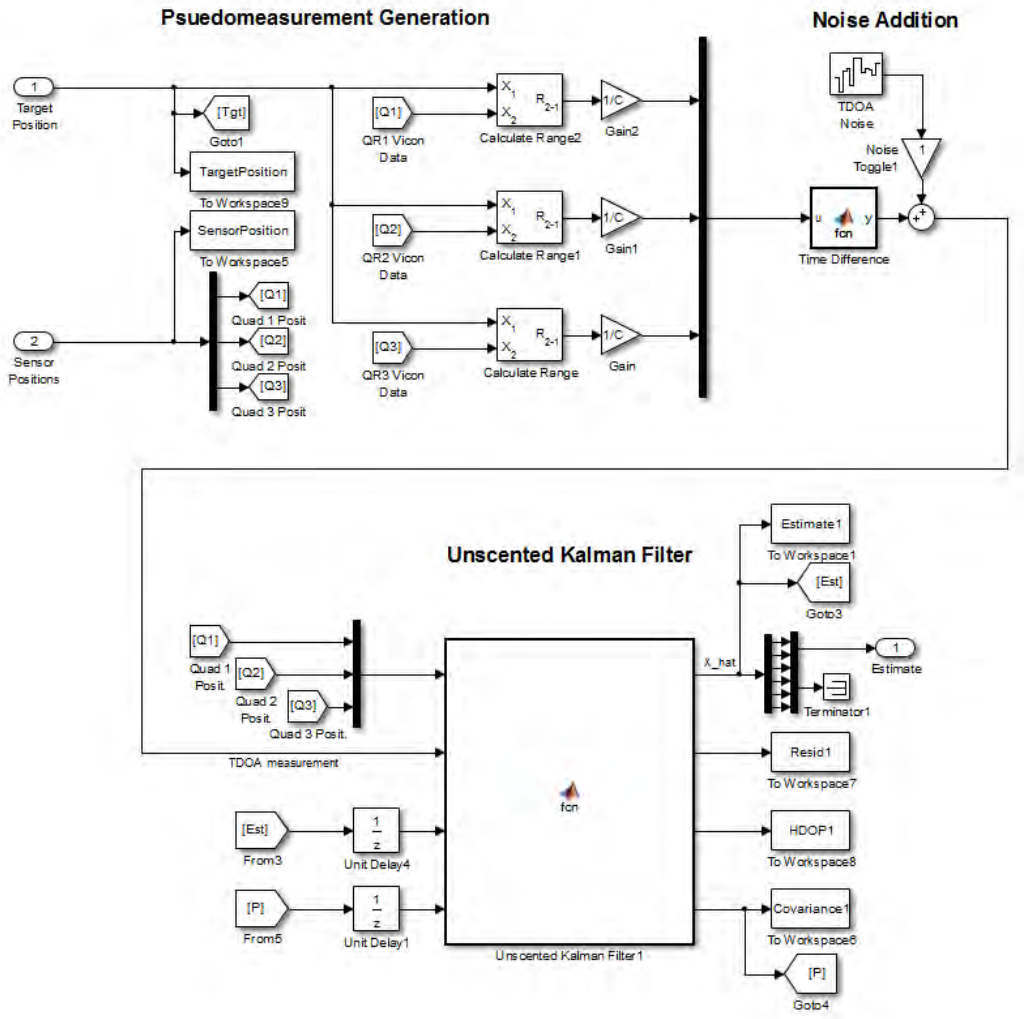

Figure 60. TDOA pseudo-measurement generation and signal routing for lab testing of the unscented Kalman filter

The structures of the TDOA pseudo-measurements that emerge from the "Time Difference" block of Figure 60, just before noise addition, take the form of Equation (V.3) for ingestion by the UKF.

$$\begin{aligned}\tau_{12} &= ArrivalTime_{Quad1} - ArrivalTime_{Quad2} \\ \tau_{23} &= ArrivalTime_{Quad2} - ArrivalTime_{Quad3} \\ \tau_{13} &= ArrivalTime_{Quad1} - ArrivalTime_{Quad3}\end{aligned} \tag{V.3}$$

The physical setup at the CAVR lab consisted of three AR.Drone quadrotors and a red helmet representing the target to be tracked. The limits of the Vicon camera studio as-positioned are approximately 8 meters by 8 meters. Yellow markers were placed on top of the quadrotor shrouds to denote the forward-looking axis of the drone. The target motion was conducted at random.

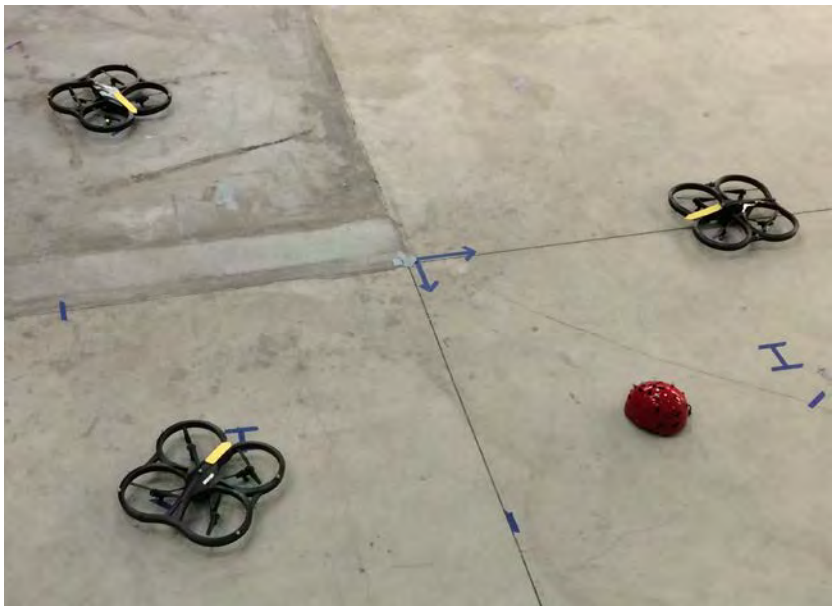

Figure 61. Physical setup of the CAVR lab test for the UKF tracking of a target (helmet, in red)

2. Unscented Kalman Filter Test Results

The lab test of the unscented Kalman filter was conducted several times and the results from one of those evaluations is detailed. To begin, the UKF was initialized with the state vector,

$$\hat{x}_o = [x, y, u, v, a_x, a_y] = [0.5, 0.5, 0, 0, 0, 0]^T \tag{V.4}$$

initial covariance,

$$P_o = diag([1e^3, 1e^3, 1e^3, 1e^3, 1e^3, 1e^3,]) \tag{V.5}$$

and noise terms,

$$Q = diag([10e^6, 10e^6, 1e^6, 1e^6, 1e^6, 1e^6])$$
$$R = diag([1e^{-6}, 1e^{-6}, 1e^{-6}]) \quad . \tag{V.6}$$

The initial estimated position in the state vector was different from the true position of the target, located at [1.2551, 0.9699], by approximately 0.9 meters. The TDOA pseudo-measurements provided to the UKF are shown in Figure 62, with the blue reference line representing the raw (true) TDOA measurement before noise addition. The mean value of the error injected via white noise was 0.0011 seconds, relating to approximately 1.7 meters when multiplied by the speed of sound.

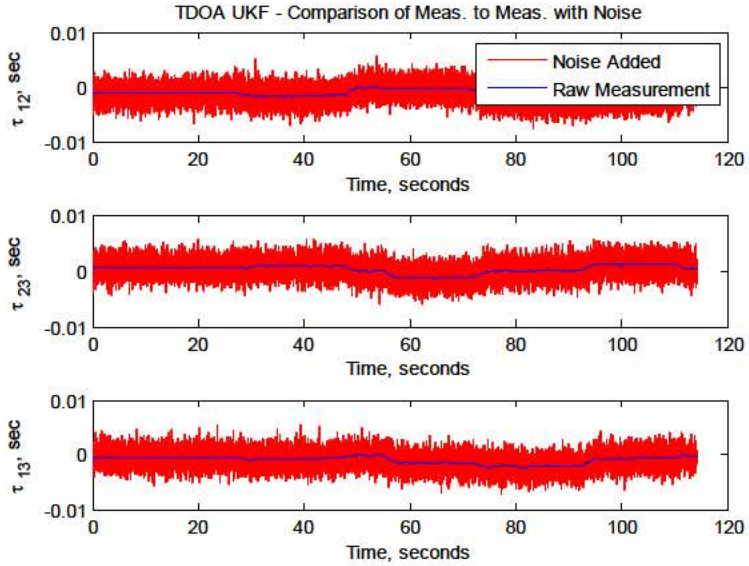

Figure 62. TDOA pseudo-measurements provided to UKF in real-time with added noise shown

The results of the test were incredibly interesting, and the overall plots are shown in Figure 63 and Figure 64. The mean square error in the position estimate was 0.21 meters and 0.88 meters for the X and Y positions, respectively. These values, however, do not tell the entire tale.

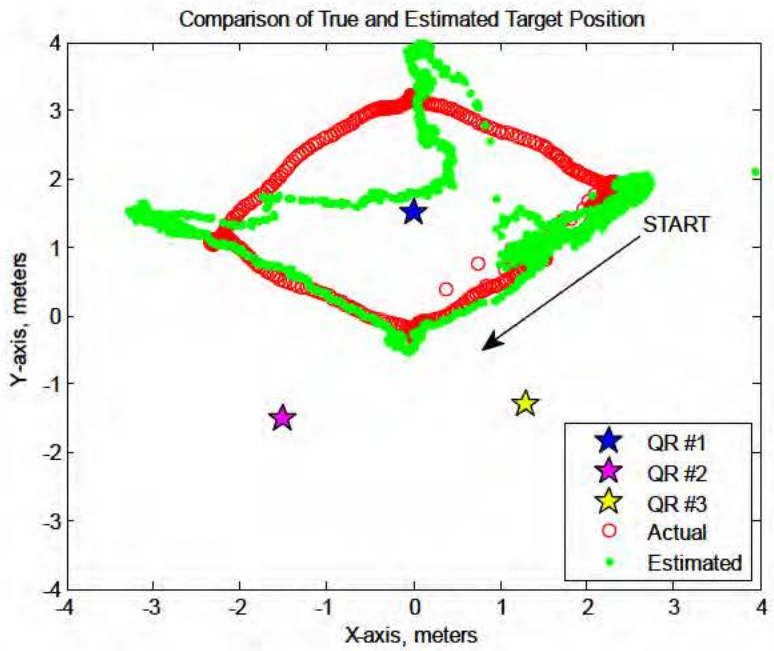

Figure 63. UKF estimated position overlaid on the actual target track

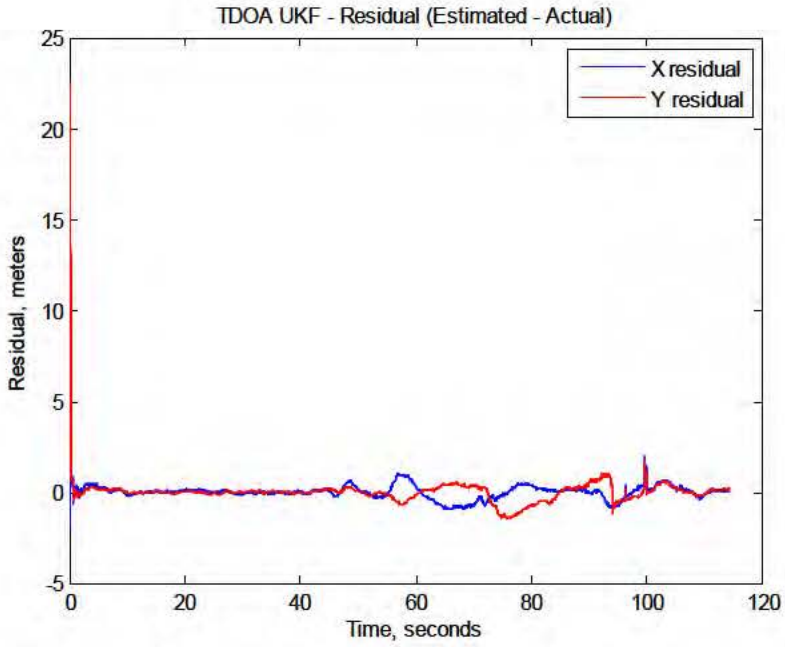

Figure 64. UKF residual in estimating position of moving target

To explore the results of this test further, it is useful to break it down into smaller time segments. When the test was first initialized, there is an expected initial overshoot as

the filter seeks convergence to the true position of the target. This is evident in the large initial residual and UKF covariance seen in Figure 65, which pertains to $t = [0,13]s$. Convergence occurs rapidly however, within two seconds, and this can also be seen in the demo screenshot of Figure 65 where all yellow markers representing the heading angle of each quadrotor are pointing towards the helmet.

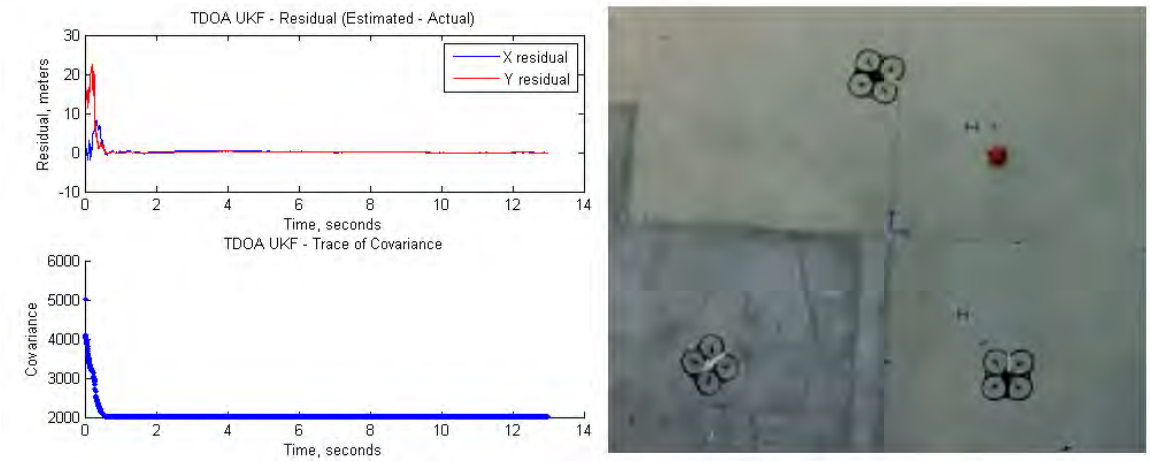

Figure 65. Initial convergence of UKF position residual and covariance with screenshot of lab demo displaying the heading of each quadrotor pointing towards the estimated position of the target

Following this convergence, the helmet is donned and the target moves towards the center of the quadrotors. Once again, as seen in Figure 66 from $t = [13,55]s$ (with the scale enlarged to show detail), there is minimal divergence in the estimated position.

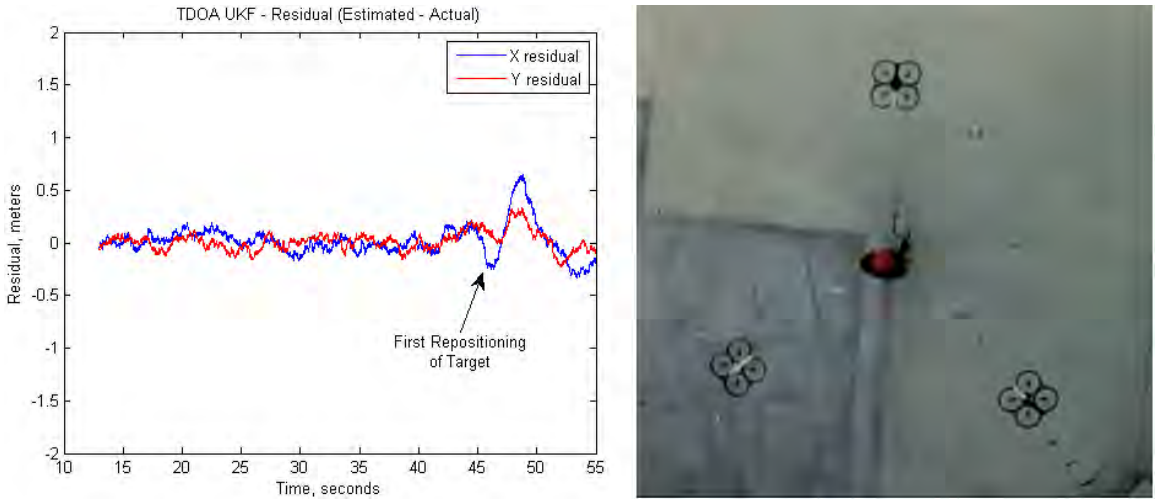

Figure 66. Continued convergence of estimated target location after first repositioning of helmet

It is after this point in time where the position estimate of the UKF begins to have larger overshoots in convergence. This error is clear in the overlay of Figure 63 and in the residuals of Figure 64 from $t = [55,120]s$, as the estimated position swings away from the true position, takes time to meet the actual position of the target and then slowly draws away.

The reason for the roughly one meter offset in residual appears to be directly tied to HDOP. Consider Figure 67, which shows HDOP and the residual as a function of equivalent timestamps. There is a direct correspondence between the peaks in HDOP and those in the residual. This further highlights the importance of including HDOP in the path-planning mission. It has a large impact on the quality of the ultimate estimated position.

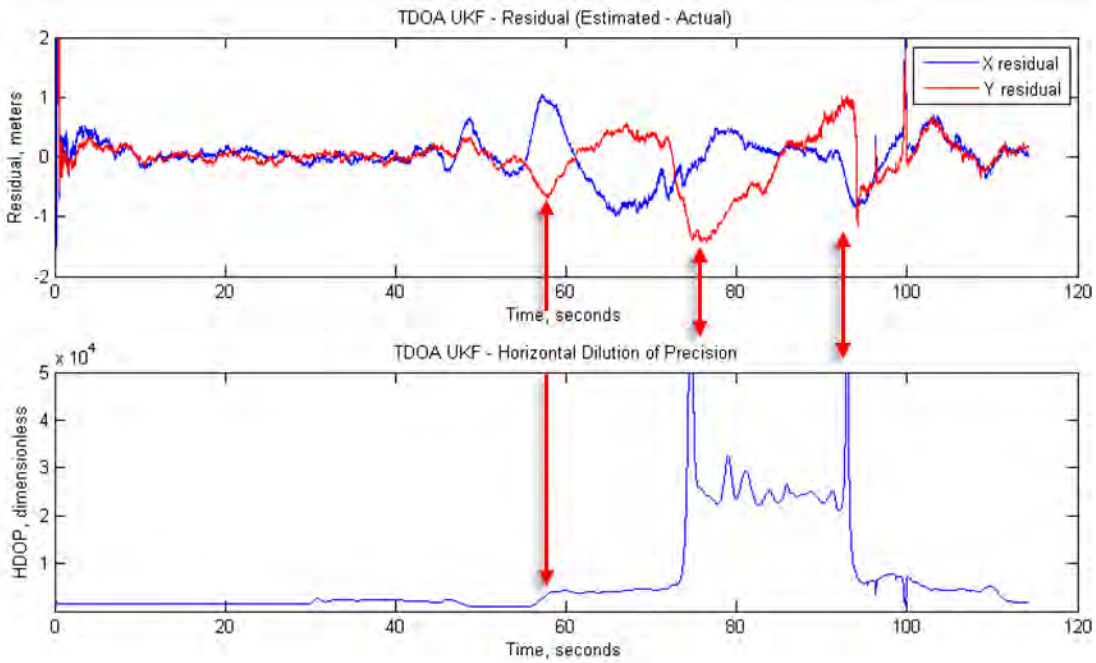

Figure 67. TDOA UKF residual peaks compared with the horizontal dilution of precision at equivalent times

Of additional note, the minimum value for HDOP in the test was observed at around the 55 second mark of Figure 67, corresponding to when the target is located near the origin. This configuration maximizes the angles between the searching quadrotors, thereby minimizing Equation (IV.13).

In general, the results of the UKF test were very encouraging. Position estimates were excellent under favorable searcher geometry. The sporadic stop-and-start nature of the target's motion presents a challenge to the estimator when HDOP is large. Once it has converged around a static position with negligible velocity, the effects of a sudden change in the position of the target (as determined via the measurements) have a characteristic rise time and overshoot, and this is exacerbated with relatively poor dilution of precision.

D. DR-RRT* PATH FOLLOWING TEST

The next test conducted in the CAVR lab was to evaluate the paths produced by the DR-RRT* algorithm and the ability of the quadrotors to follow them. Again, we

present the envisioned at-sea scenario that the test is founded upon, provide some relevant implementation details, and discuss the results.

1. DR-RRT* Path-Following Scenario

We considered the scenario of two AquaQuads that are tasked to reposition themselves in order to maintain an ostensibly favorable initial geometric configuration. The vortex ocean current pattern that we have utilized throughout this thesis continuously draws the AquaQuads towards the center in simulation, making potential station-keeping in this environment expensive from an energy consumption standpoint. Therefore, the AquaQuads are directed to swap positions by following DR-RRT* paths.

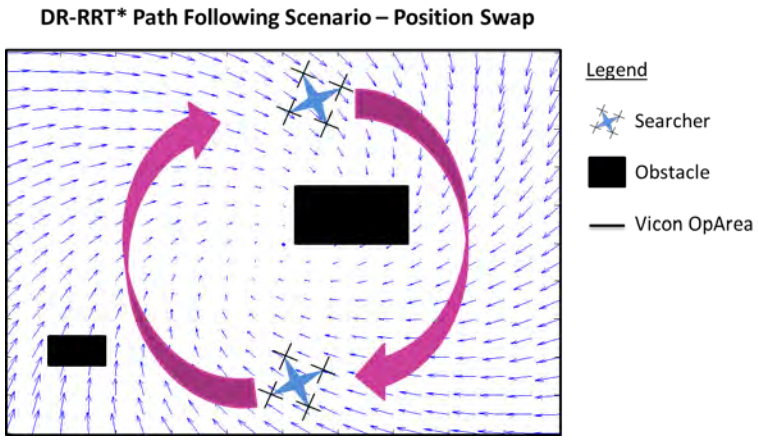

Figure 68. Position swapping scenario for the DR-RRT* path following test with ocean current vectors used to program simulated drifting behavior into the AR.Drone quadrotors

In order to approximate the drifting and hopping behavior of the prototype AquaQuads, it was necessary to make some modifications to the "Trajectory Planner" block of Figure 58. Initial attempts at forcing the AR.Drone quadrotors to drift in air currents generated by a fan were hampered by the proprietary Parrot software that includes disturbance rejection when in a hovering state (i.e., the AR.Drone actively fought against the air flow). Drifting was instead simulated with a "rabbit-following" technique. At a 1Hz frequency, the position of the quadrotor was used to define the ocean current in its vicinity via a lookup table. This ocean current velocity was used to create a

waypoint, which was placed at one dead-reckoned time period away. At the next period, the position of the quadrotor was evaluated to determine if it fell within a watch circle radius around the waypoint; if so, another waypoint was generated in the same manner. While following a drift path, the CAVR quadrotors were commanded to fly at an altitude of one meter. When hopping, its altitude was raised to two meters and the quadrotor was allowed to rapidly fly to the terminal location of the hop. In this way, the hops were immediately visible to an observer.

At this point it is important to note that, like the envisioned AquaQuad implementation it is based upon, the drift behavior of the AR.Drone is independent of the DR-RRT* path. We place the quadrotor at the starting location of the DR-RRT* tree, after which errors from external disturbances and regional ocean current approximations are free to build. This is expected from a drifting platform. Path-specific position commands are only provided to the quadrotor when it reaches a hopping location. Flight steps that terminate at a defined location act to reset the drifting errors.

The map used within the DR-RRT* algorithm and the trajectory planner block is shown in Figure 69 alongside of a screenshot of the physical quadrotors in their starting locations for the video recorded demo.

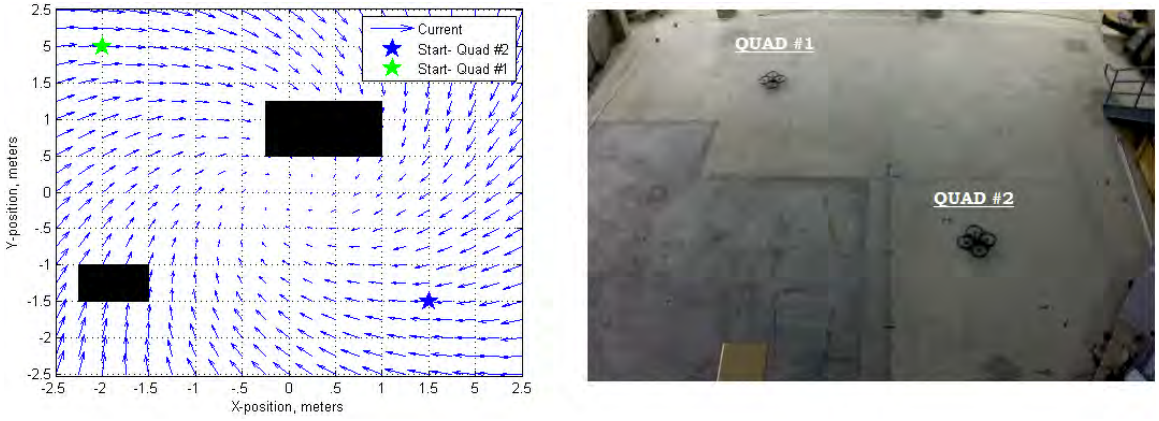

Figure 69. Initialization phase of the DR-RRT* path-following lab test

2. DR-RRT* Path-Following Test Results

The same DR-RRT* algorithm was run for each quadrotor simultaneously, incorporating the initialization plot of Figure 69 and considering the starting position of one the goal point of the other. Once a successful path to the goal was found, the script terminated. This resulted in the trees of Figure 70, with Quad #1 on the left and Quad #2 on the right.

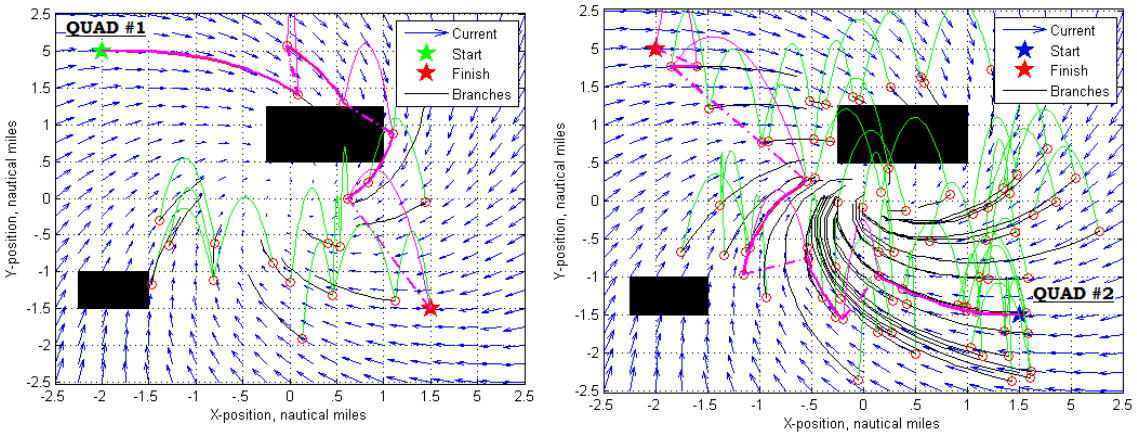

Figure 70. Result of DR-RRT* algorithm run for two quadrotors swapping positions in the presence of simulated ocean current and an obstacle field

It was anticipated that this scenario would result in opposing trees with similar shapes, driven by the symmetry of the ocean current in which they operate. This was indeed the case, as both trees find drift paths towards the origin and simply fly to the finish once the opposing ocean current is no longer beneficial. Figure 71 removes all of the branches analyzed in the tree for clarity and presents the final path that each quadrotor must follow.

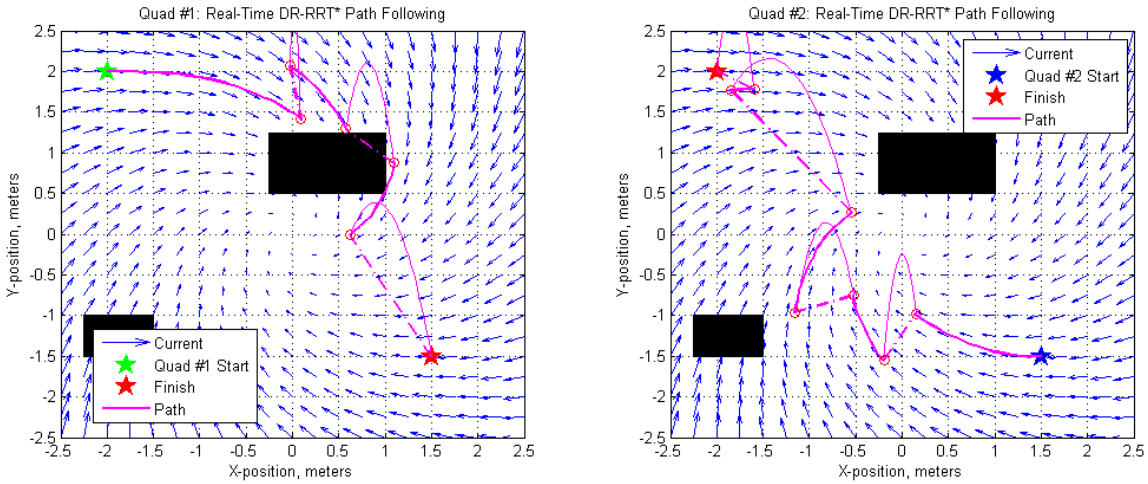

Figure 71. Final path from DR-RRT* algorithm run for two quadrotors swapping positions in the presence of simulated ocean current and an obstacle field

With the paths so designated, the quadrotors in the lab were directed to follow them. Figure 72 displays the overlay of each quadrotor's position along the path sampled at 1Hz by the Vicon system represented as red plus signs. The overarching result of the test is that both quadrotors made it to their respective goal successfully by following the DR-RRT* path. Figure 73 shows two screenshots of the test illustrating hopping segments. The astute observer may note that Quad #1's Vicon position in Figure 72 occasionally falls within the limits of the obstacle, despite the fact the DR-RRT* path does not. This is a common issue with RRT's in that they often consider the path-follower to be a point mass with negligible dimension. A simple solution to this problem is to enlarge the obstacle size by a safety factor (not shown) in the planning process. One additional note is that the gaps in quadrotor position that visibly correspond to hop segments are in fact a function of the increased speed of the quadrotors during these segments, resulting in far fewer 1Hz samples for plotting. Therefore some of the position updates in Figure 72 which fall either on obstacles or in mid-hop do not necessarily represent erroneous drifting behavior.

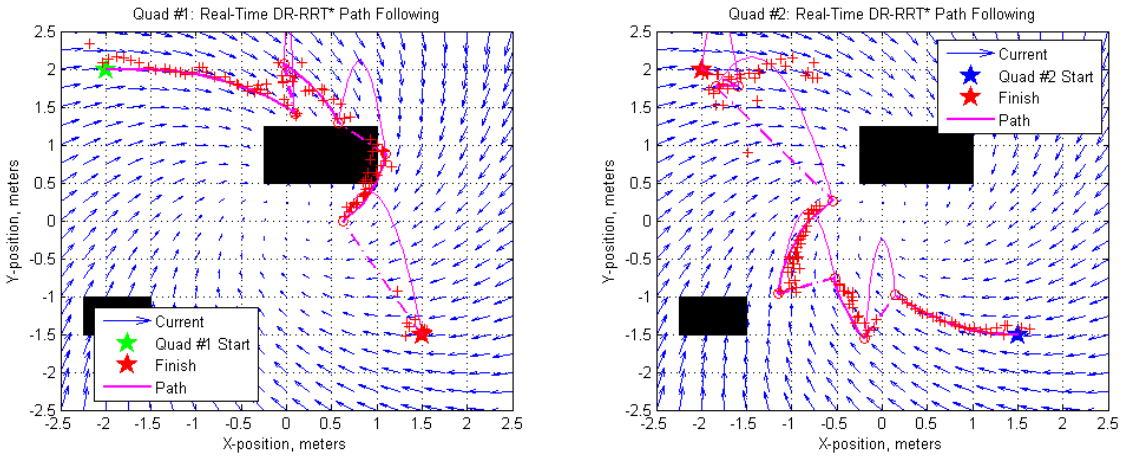

Figure 72. Plotted position overlay of Quad #1 and #2 following at DR-RRT* generated path in the lab test

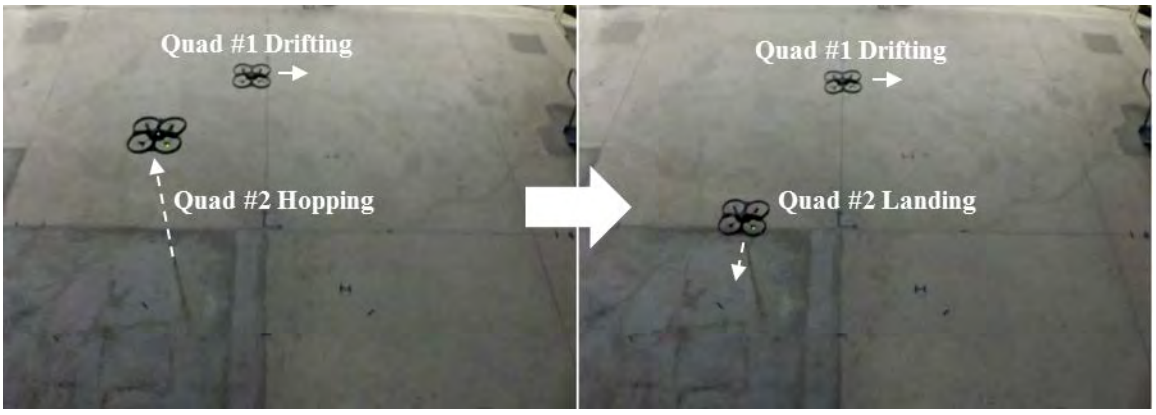

Figure 73. Screenshot of DR-RRT* path-following lab test showing a change in altitude indicative of a hopping segment

To explore the result of the test further, we focus upon Quad #2, as it highlights some interesting behavior. Figure 74 shows the initial path taken by Quad #2. Again, gaps in position markers are a visual indication of hopping behavior. This is useful for the reader, but is a byproduct of the relatively short flight length as compared to the sampling frequency for position.

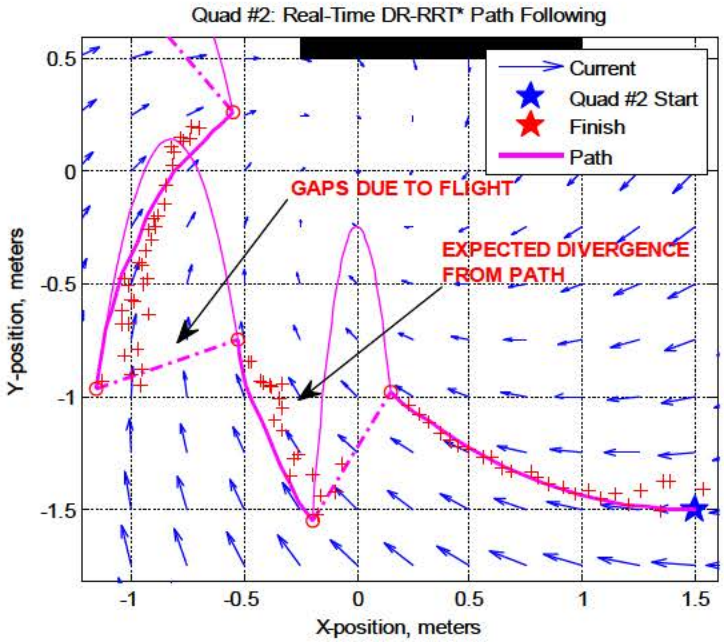

Figure 74. Close up view of the actual position of Quad #2 along the DR-RRT* path during the lab test

It is evident from Figure 74 that the position of Quad #2 does not exactly match the path planned by the DR-RRT*. This behavior is precisely what we anticipated. The paths were created assuming a perfect dead-reckoned solution in simulation. However, even in our relatively disturbance-free lab environment, there was still a noticeable effect from interaction between the propeller wash of each quadrotor. In addition, the location of the AR.Drone after hopping was unregulated with respect to the drifting segments of the DR-RRT* path; therefore the quadrotor was free to drift in whatever current vector it found itself in. Exactly like its envisioned real-world implementation.

At every sampling interval, the trajectory planner block of Figure 58 would check the position of the quadrotor and determine if it fell within a watch circle radius of a hopping location. If so, it would execute the hop segment. The size of the watch circle must be designed with the expected drifting error in mind. If the circle is too large, excessive flight time and energy expenditure will be the result. If the circle is too small, the drifter may miss it and stray far from the path, forcing the platform to re-plan a new path. In light of our energy focus, the conservative approach is to err towards a smaller watch circle at the cost of potential re-planning.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

In this thesis, we provide an overview of the prototype AquaQuad platform and the capabilities it is being built to possess as a major force multiplier. Driving the purpose of the AquaQuad is to perform target position estimation, and the quality of the flock's solution will directly impact either weapon effectiveness for submarine prosecution or behavior analysis for marine mammal research. Meanwhile, the goal of providing these estimates autonomously over a large time scale is fundamentally and near-universally limited by the existing battery capacity available to AUVs. These aspects of the envisioned mission motivated the efforts contained in this work to design and evaluate two major algorithms that support the AquaQuad in the conduct of sustained, energy-efficient surveillance.

A. UNSCENTED KALMAN FILTER ALGORITHM

The first contribution adapted the existing unscented Kalman filter (UKF) framework to the task of estimating the state of a target utilizing distributed nonlinear measurements. The UKF was created, simulated and evaluated for quality incorporating several different measurement types. The result of this analysis displayed its excellent performance characteristics. In a bearing-only tracking scenario there was a near order of magnitude difference in mean square position error as compared to the extended Kalman filter.

Within the framework of nonlinear estimation, the horizontal dilution of precision (HDOP) was explored as a metric for which to base the precision available to the estimator as a function of sensor geometric positioning. The value of HDOP was seen to be dependent upon range and orientation; however the specific proportionality was shown to be tied directly to the measurement type.

The UKF was then implemented in real-time on quadrotors in the Center for Autonomous Vehicle Research (CAVR) laboratory to track a target undergoing non-uniform motion. The position estimate of the real-time UKF was excellent under favorable geometry, while unfavorable (high HDOP) conditions resulted in greater filter

convergence time to the true position of the target. This behavior highlighted the importance of regulating the position of the sensor platforms to accomplish group objectives.

Limitations on the results of our UKF implementation exist due to expected variation in filter performance depending upon the values chosen by the user for process, measurement, and initial Kalman filter covariance. The measurement covariance in particular relates to actual sensor performance, which is controlled in our simulation and testing through the use of pseudo-measurements. In addition, these pseudo-measurements are provided at a consistent frequency, whereas acoustic emissions from targets are expected to be intermittent.

B. DR-RRT* ALGORITHM

The second contribution of this thesis built upon current sampling-based methods in the construct of a new and novel tool for path-planning: the Dead-Reckoning Rapidly-Exploring Random Tree Star (DR-RRT*) algorithm. The DR-RRT* takes the infinite-time optimality guarantees of the RRT* and incorporates limitations on AquaQuad control authority designed to reduce expenditure of stored energy.

A Monte Carlo experiment was conducted to develop objective statistics for the energy that was consumed using DR-RRT* paths. The result was a mean energy savings of 22 percent over the energy that would have been expended flying directly to the goal. When the DR-RRT* was allowed to run beyond its initial solution, that savings increased to 27 percent at the expense of several seconds of additional computation time. Each of the paths that were created was furthermore guaranteed to be obstacle-free and incorporated an evaluation of HDOP at every branch to reduce the occurrence of poor geometry with respect to the nonlinear estimation process.

Finally, the paths generated by the DR-RRT* were successfully followed by two flying quadrotors in the CAVR lab. Each programmed to simulate drifting behavior in the air; the quadrotors exchanged initial positions with a series of unregulated drifts and directed hops as projected in the path provided to them.

Limitations of the DR-RRT* method will be based upon the assumptions made in the creation of the paths: most importantly the ocean current distribution that the algorithm is initialized with. Live updates of local current flow and periodic re-planning can assist in minimizing the error between prediction and reality.

C. FUTURE WORK

In our work, the concept of harvesting solar energy is presented using a 24-hour average of solar radiation as a baseline. To ensure persistence in the battle space, a more complex representation of available solar energy must be used based upon location, angle of incidence, and most importantly time of day. These variables can be incorporated into the DR-RRT* planning framework to better approximate the expected energy balance. A global flock planner can then be designed that takes into account the projected battery life of each AquaQuad and selects elements for repositioning based upon it.

This thesis has also briefly detailed the signal processing approaches used to obtain measurements of a target's location in the ocean. The actual sensor employed and the specific signal processing techniques utilized must be constructed and tested for implementation on the AquaQuad. The UKF code provided can then be tuned for the sensor in use, adjusted to account for gaps in time between measurements and fielded in an aquatic environment.

Physical construction of the AquaQuad is ongoing, and as such there is much work to be done in the following realms:

1. Propulsion

Optimization of the motor and propeller combination can greatly increase the performance of the quadrotor. The ocean environment is dynamic and highly corrosive; therefore improvements to prevent long and short-term fouling of exposed parts will have a large effect upon longevity of the platform at sea.

2. Controls

Employment of the sensor beneath the AquaQuad will produce varying forces and moments when repositioning. Modified control laws for slung loads can be investigated to account for this.

3. Power Conversion and Storage

Battery selection is being finalized, yet integration of the solar array on board the AquaQuad is incomplete. Studies regarding battery life-cycle analysis, photovoltaic device maximum power point tracking (MPPT), and surface treatments for antifouling of the solar cells can be conducted.

APPENDIX A. ACOUSONDE TECHNICAL SPECIFICATIONS

The *Acousonde*™ is a self-contained underwater acoustic recorder comprising one or, optionally, two hydrophones, sensors for attitude, orientation, depth and temperature, a digital recorder, and a field-replaceable battery. Attached to a subject with suction cups or other means, the Acousonde measures the subject's sound environment as well as potentially associated behavior.

Lori Mazzuca

Ward Testa

In addition to its primary mission as a tool for assessing the impact of noise on marine wildlife, the Acousonde can be used to study vocalization behavior of the tagged subject. The instrument may also be applied as an autonomous recorder suspended from a cable, placed on the seafloor, or housed in a robotic or remotely-operated vehicle.

Lori Mazzuca

SPECIFICATIONS, ACOUSONDE 3A UNDERWATER ACOUSTIC RECORDER

Maximum operating depth (fixed build option)	500 m (-500m suffix) / 1000 m (-1km) / 2000 m (-2km) / 3000 m (-3km)
Maximum continuous acoustic sampling rate	232 kHz
Anti-alias filter, low-power (LP) channel	8-pole elliptic, adjustable (automatic) up to 9.2 kHz maximum
Anti-alias filter, high-frequency (HF) channel	6-pole linear phase, fixed
3-dB anti-alias cutoff	9.2 kHz (LP chan max); 42 kHz (HF chan)
22-dB anti-alias cutoff	11.1 kHz (LP chan max); 100 kHz (HF chan)
3-dB high-pass cutoff	22 Hz (LP chan); 20 Hz/1 kHz/10 kHz (HF chan, fixed, customer spec)
Unamplified raw ceramic sensitivity, re 1 V/µPa	-201 dB (LP chan hydrophone) & -204 dB (HF chan hydrophone)
Saturation at 0-dB gain, re 1 µPa zero-peak	187 dB (LP chan) & 176 dB (HF chan)
Acoustic gains, selectable at deployment	0 or +20 dB
Acoustic sampling resolution	16 bits
Auxiliary sampling rate	Up to 800 Hz (3D tilt), 40 Hz (3D compass), 10 Hz (depth, temp)
Auxiliary sampling resolution	16 bits (except 10 bits for tilt)
Auxiliary sampling channels	Depth (pressure), internal temperature, 3D tilt, 3D compass
Total storage capacity (primary & spare)	64 GB, 128 GB max (*at sample rates < 26 kHz, battery limits storage*)
Maximum duration if sampling < 26 kHz	6-14 days depending on temperature and if aux sampling also active
Maximum measured data download rate	3.3 GB/hour via MicroUSB connector

December 2013

Firmware support for some specifications, performance and/or functionality may be pending; see current release notes. Data subject to change without notice.

HOME PAGE http://www.acousonde.com
TECH QUESTIONS tech@acousonde.com
SALES acousonde@cetaceanresearch.com

The Acousonde™ is made by Acoustimetrics, a brand of Greeneridge Sciences, Inc.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. UNSCENTED KALMAN FILTER MATLAB CODE

```
function [Estimate, Residual, HDOP, NewP] =
fcn(SensorPosition,Measurement,PriorEstimate,PriorP)

%========INFO========
%LT Chase Dillard
%UKF Code for Bearing-Only Tracking
%11/26/2014

%Implemented in SIMULINK within a MATLAB function block
%Version: MATLAB R2014A
%Based upon:  E. A. Wan and R. van der Merwe,
%        "The Unscented Kalman Filter for Nonlinear
%        Estimation," IEEE, 2000.

%Inputs:  SensorPosition - (x,y) position of the 3 bearing sensors
%              Format: [x1,x2,x3,y1,y2,y3]
%      Measurement - bearing to target in radians
%      PriorEstimate - previous UKF state estimate (req's unit delay)
%      PriorP - previous UKF covariance (req's unit delay)

%Outputs:  Estimate - UKF estimate of target's (x,y,u,v,ax,ay)'
%      Residual - Delta b/w expected and actual measurement. Optional.
%      HDOP - Horizontal dilution of precision for BOT. Optional.
%      NewP - updated UKF covariance
%====================


%UKF Parameters
n=length(PriorEstimate);        %number of states
deltat = 0.005;             %time step, seconds
Q = 1e-6*diag([10 10 1 1 1 1]);    %process noise
R = diag([0.001^2, 0.001^2, 0.001^2]); %measurement noise

%Conduct UKF
L=numel(PriorEstimate);       %number of states
m=numel(Measurement);        %number of measurements
alpha=1e-3;              %spread of the sigma points
ki=0;                %second scaling parameter ~0
beta=2;               %prob dist of x, Gaussian
lambda=alpha^2*(L+ki)-L;      %scaling factor
c=L+lambda;              %scaling factor
Wm=[lambda/c 0.5/c+zeros(1,2*L)];  %weights for means
Wc=Wm;
Wc(1)=Wc(1)+(1-alpha^2+beta);    %weights for covariance
c=sqrt(c);

%Determine sigma points
P = PriorP;
A = c*chol(P)';
Y = PriorEstimate(:,ones(1,numel(PriorEstimate)));
```

```
X = [PriorEstimate Y+A Y-A];        %sigma points around x

%Unscented transformation of process
L=size(X,2);
x1=zeros(n,1);
X1=zeros(n,L);
for k=1:L % propagation of sigma points through process fcn
  X1(:,k)=X(:,k)+[X(3,k)*deltat; X(4,k)*deltat; 0; 0; 0; 0];
  x1=x1+Wm(k)*X1(:,k);          % weighted average
end
X2=X1-x1(:,ones(1,L));
P1=X2*diag(Wc)*X2'+Q;            % weighted outer-product

%Unscented transformation of measurements
L=size(X1,2);
z1=zeros(m,1);
Z1=zeros(m,L);
for k=1:L % propagation of sigma points through nonlinear bearing fcn
  Z1(:,k)=atan2(X1(2,k)-SensorPosition(4:6),X1(1,k)-
SensorPosition(1:3));
  z1=z1+Wm(k)*Z1(:,k);          % weighted average
end
Z2=Z1-z1(:,ones(1,L));
P2=Z2*diag(Wc)*Z2'+R;            % weighted outer-product

%Final Calculations
P12=X2*diag(Wc)*Z2';           %transformed cross-covariance
K=P12*inv(P2);               %Kalman gain calculation
Residual = Measurement-z1;         %Residual for plotting
x=x1+K*(Measurement-z1);          %state update
P=P1-K*P12';                %covariance update

%Parameters for dilution of precision calculation
R1=(x(2)-SensorPosition(4))^2+(x(1)-SensorPosition(1))^2;
R2=(x(2)-SensorPosition(5))^2+(x(1)-SensorPosition(2))^2;
R3=(x(2)-SensorPosition(6))^2+(x(1)-SensorPosition(3))^2;

%H matrix for HDOP calculation
H_HDOP = [-(x(2)-SensorPosition(4))/R1, (x(1)-SensorPosition(1))/R1;
     -(x(2)-SensorPosition(5))/R2, (x(1)-SensorPosition(2))/R2;
     -(x(2)-SensorPosition(6))/R3, (x(1)-SensorPosition(3))/R3];

%Outputs
HDOP = sqrt(trace(inv(H_HDOP'*H_HDOP)));
NewP = P;
Estimate = x;
Resid = Residual;
```

```
function [Estimate, Residual, HDOP, NewP] =
fcn(SensorPosition,Measurement,PriorEstimate,PriorP)

%========INFO========
%LT Chase Dillard
%UKF Code for Time-Difference of Arrival Tracking
%11/26/2014

%Implemented in SIMULINK within a MATLAB function block
%Version: MATLAB R2014A
%Based upon:  E. A. Wan and R. van der Merwe,
%        "The Unscented Kalman Filter for Nonlinear
%        Estimation," IEEE, 2000.

%Inputs:  SensorPosition - (x,y) position of the 3 TDOA sensors
%              Format: [x1,y1,x2,y2,x3,y3]
%      Measurement - TDOA measurement in seconds, specifically:
%              Tau12 (Diff b/w arrival time of Sensor 1 and 2)
%              Tau23 (Diff b/w arrival time of Sensor 2 and 3)
%              Tau13 (Diff b/w arrival time of Sensor 1 and 3)
%      PriorEstimate - previous UKF state estimate (req's unit delay)
%      PriorP - previous UKF covariance (req's unit delay)

%Outputs:  Estimate - UKF estimate of target's (x,y,u,v,ax,ay)'
%      Residual - Delta b/w expected and actual measurement. Optional.
%      HDOP - Horizontal dilution of precision for TDOA. Optional.
%      NewP - updated UKF covariance
%====================

%UKF Parameters
n=length(PriorEstimate);        %number of states
deltat = 0.005;             %time step, seconds
Q = 1e-6*diag([10 10 1 1 1 1]);    %process noise
R = diag([0.001^2, 0.001^2, 0.001^2]); %measurement noise
C = 1500;                %Ocean Sound Speed, m/s

%Conduct UKF
L=numel(PriorEstimate);         %number of states
m=numel(Measurement);          %number of measurements
alpha=1e-3;               %spread of the sigma points
ki=0;                  %second scaling parameter ~0
beta=2;                 %prob dist of x; Gaussian
lambda=alpha^2*(L+ki)-L;        %scaling factor
c=L+lambda;               %scaling factor
Wm=[lambda/c 0.5/c+zeros(1,2*L)];    %weights for means
Wc=Wm;
Wc(1)=Wc(1)+(1-alpha^2+beta);      %weights for covariance
c=sqrt(c);

%Determine sigma points
P = PriorP;
A = c*chol(P)';
Y = PriorEstimate(:,ones(1,numel(PriorEstimate)));
X = [PriorEstimate Y+A Y-A];        %sigma points around x
```

```
%Unscented transformation of process
L=size(X,2);
x1=zeros(n,1);
X1=zeros(n,L);
for k=1:L %propagation of sigma points through process fcn
  X1(:,k)=X(:,k)+[X(3,k)*deltat; X(4,k)*deltat; 0; 0; 0; 0];
  x1=x1+Wm(k)*X1(:,k);          % weighted average
end
X2=X1-x1(:,ones(1,L));
P1=X2*diag(Wc)*X2'+Q;            % weighted outer-product

%Unscented transformation of measurements
L=size(X1,2);
z1=zeros(m,1);
Z1=zeros(m,L);
for k=1:L %propagation of sigma points through nonlinear TDOA fcn
  Z1(:,k)=(1/C)*[sqrt((X1(1,k)-SensorPosition(1))^2+(X1(2,k)-
SensorPosition(2))^2)-sqrt((X1(1,k)-SensorPosition(3))^2+(X1(2,k)-
SensorPosition(4))^2); %Time delay b/w buoys 1 & 2
          sqrt((X1(1,k)-SensorPosition(3))^2+(X1(2,k)-
SensorPosition(4))^2)-sqrt((X1(1,k)-SensorPosition(5))^2+(X1(2,k)-
SensorPosition(6))^2); %Time delay b/w buoys 2 & 3
          sqrt((X1(1,k)-SensorPosition(1))^2+(X1(2,k)-
SensorPosition(2))^2)-sqrt((X1(1,k)-SensorPosition(5))^2+(X1(2,k)-
SensorPosition(6))^2)]; %Time delay b/w buoys 1 & 3
  z1=z1+Wm(k)*Z1(:,k);          % weighted average
end
Z2=Z1-z1(:,ones(1,L));
P2=Z2*diag(Wc)*Z2'+R;            % weighted outer-product

%Final Calculations
P12=X2*diag(Wc)*Z2';            %transformed cross-covariance
K=P12*inv(P2);                %Kalman gain calculation
Residual = Measurement-z1;          %Residual for plotting
x=x1+K*(Measurement-z1);          %state update
P=P1-K*P12';                %covariance update


%Calculate range between target estimate and sensors for TDOA HDOP
R1T=sqrt((x(2)-SensorPosition(2))^2+(x(1)-SensorPosition(1))^2);
R2T=sqrt((x(2)-SensorPosition(4))^2+(x(1)-SensorPosition(3))^2);
R3T=sqrt((x(2)-SensorPosition(6))^2+(x(1)-SensorPosition(5))^2);

%H matrix for TDOA HDOP calculation
H_HDOP = 1/C*[(SensorPosition(3)-x(1))/R2T - (SensorPosition(1)-
x(1))/R1T, (SensorPosition(4)-x(2))/R2T - (SensorPosition(2)-x(2))/R1T;
       (SensorPosition(5)-x(1))/R3T - (SensorPosition(3)-x(1))/R2T,
(SensorPosition(6)-x(2))/R3T - (SensorPosition(4)-x(2))/R2T;
       (SensorPosition(5)-x(1))/R3T - (SensorPosition(1)-x(1))/R1T,
(SensorPosition(6)-x(2))/R3T - (SensorPosition(2)-x(2))/R1T];

%Outputs
HDOP = sqrt(trace(inv(H_HDOP'*H_HDOP)));
```

```
NewP = P;
Estimate = x;
Resid = Residual;
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. DEAD-RECKONING RAPIDLY-EXPLORING RANDOM TREE STAR CODE

```
%========INFO========

%LT Chase Dillard
%Dead-Reckoning Rapidly-Exploring Random Tree Star (DR-RRT*) Algorithm
%11/26/2014

%Requires the following custom MATLAB functions (included in Appendix):
%        HDOP_Calc.m
%        DR_RRTstar_Extend.m
%        CellFinder.m

%Implemented in MATLAB. Version: MATLAB R2012b
%Based upon:  S. Karaman and E. Frazzoli, "Optimal kinodynamic motion
%        planning using incremental sampling-based methods," in
%        49th IEEE Conference on Decision and Control,
%        Atlanta, GA, 2010.

%====================

%% Create map space
clc;
clear all;
clf;

% Produce ocean current parameters and create map
[x1, x2] = meshgrid(-.5:0.05:0.5, -.5:.05:.5); %Sampling mesh of points
x1d = -x1 - 2 *x2 .*x1.^2+x2;          %Equations defining vector field
x2d = -x1-x2;
SF = 10000;                    %Scaling Factor for map adjustment
x1 = SF.*x1; x2 = SF.*x2;             %Scaling up map space
figure(1)                     %Plot the vector field
quiver(x1,x2,x1d, x2d,1,'Color','b');
title('Dead Reckoning – Rapidly-Exploring Random Tree Star (DR-RRT*)
Algorithm in a Current Field')
hold on;
axis tight;

% Other map parameters
GridSpace = length(x1);                %One-sided dimension of grid
space
ObstacleSpace = zeros(GridSpace,GridSpace);       %Establish obstacle
space grid for later population
MapLength = 0.5*(x1(GridSpace, GridSpace) - x1(1,1));  %Half of one-
sided dimension of X
xlimits = [x1(1,1) x1(GridSpace, GridSpace)];      %X limits of map
ylimits = [x2(1,1) x2(GridSpace, GridSpace)];      %Y limits of map
MaxDistance = sqrt((2*MapLength)^2+(2*MapLength)^2);  %Max distance
possible for AquaQuads to travel. Used in cost fcn.
```

```
% Generate and plot obstacles
ObstaclePlot=[];
try
  for i=1:2;
    xObs = round(abs(GridSpace.*rand(1)));       %X-coordinate of
random obstacle
    yObs = round(abs(GridSpace.*rand(1)));       %Y-coordinate of
random obstacle
    wObs= round(abs(0.2*GridSpace.*rand(1)));     %Width of random
obstacle
    hObs= round(abs(0.2*GridSpace.*rand(1)));     %Height of random
obstacle
    ObstaclePlot = [ObstaclePlot; xObs yObs wObs hObs]; %Maintaining
obstacle position for later plotting
    ObstacleSpace(yObs:yObs+hObs,xObs:xObs+wObs)=1;   %Filling
obstacles spaces with "1"
    rectangle('Position', [-0.55*SF+(xObs)*0.05*SF 0.55*SF-
(yObs+hObs)*0.05*SF (wObs+1)*0.05*SF (hObs+1)*0.05*SF], 'FaceColor',
'black'); %Plotting rectangle
  end
catch
  display('Error producing obstacles. Try again.')
end


%% RRT Parameters

% Establish Standard User-Defined RRT Parameters
iterations = 10000;       %Establish desired number of iterations
Start = [-4*SF/10 4*SF/10];   %Start Point
Finish = [3*SF/10 -3*SF/10];  %Final Goal Point
GoalProbability = 0.05;     %Forces selection of Goal Point as Target
with a fixed probabilty
distThresh = 100;        %Maximum distance to define success at goal
N = 5;             %Number of Nearest nodes to Target to evaluate
dt = 60;            %Time delta to integrate kinematics across, seconds
RewireRadius = SF/4;      %Radius to look for nearest neighbors in
rewire step

% Establish Other RRT Parameters
Tree = [1 Start 1 0 0 0];                %Establishes Tree as [Node#,
x, y, Parent#, Branch Cost, Cumulative Cost, HopIndicator]
MaxFlightTime = 2;                   %Max desired flight time, minutes;
HopDistance = (20*2000)*(MaxFlightTime/60);       %Maximum hopping
distance = (20 knots * X hours of flight), meters.
MaxHDOP = 9000;                     %Max HDOP possible in scenario.
Used in cost fcn. ASSUMED.
Quad2 = [5000 5000]; Quad3 = [5000 -5000];       %Positions of other
players on the perimeter
Quad4 = [-5000 -5000]; Sensors = [Quad2 Quad3 Quad4];
Sub = [(Finish(1) + 100) (Finish(2) + 100)];      %Sub position offset
to prevent singularity at Goal
```

```
plothandle = [];                    %Handle for line plots of each
vertex. Enables deletion of vertices during Rewire.


% Plot initial and terminal positions
hold on
plot(Start(1),Start(2),'gp','MarkerSize',14,'MarkerFaceColor','g')
hold on
plot(Finish(1),Finish(2),'rp','MarkerSize',14,'MarkerFaceColor','r')
set(gca,'XTickLabel',{'-2.5','-2','-1.5','-1','-
.5','0','.5','1','1.5','5','2.5'})
set(gca,'YTickLabel',{'-2.5','-2','-1.5','-1','-
.5','0','.5','1','1.5','5','2.5'})
xlabel('X-position, nautical miles')
ylabel('Y-position, nautical miles')
grid on

%Cost function
% eta_solar = 0.24;   %Solar cell efficiency, percent
% array_area = 0.3;   %Solar cell area on AquaQuad, meters
solar_energy = 400;   %Solar energy, calc'd offline as
(AvgRadiation*efficiency*array_area), Watt-hours in a 24-hr period
flight_power = 200;   %Power required for flight, Watt
FlightEnergyMax = flight_power*(MaxFlightTime/60); %Energy req'd for
maximum time of flight
SolarEnergyMax = solar_energy/24/3600; %Maximum solar energy possible
every second. NOTE: THIS ESSENTIALLY UNDOES THE WHOLE POINT OF THIS
CALCULATION

%Cost function gain terms
K1 = 0.0225;       %Associated term is normalized HDOP
K2 = 0.0225;       %Associated term is normalized Dist2Goal
K3 = 0.95;        %Associated term converts 200W power used for 2 min
of flight. Gain makes this term "0.9" to dominate
K4 = 0.005;       %Associated term is solar energy

%% Run DR-RRT* Loop

tic
for i = 1:iterations;

  % Dead reckon position
  [Cell, V, vx, vy] = CellFinder(Tree(end,2:3), x1, x2, x1d, x2d);
  NewBranch = DR_RRTstar_Extend(xlimits, ylimits, Tree(end,2:3), 'DR',
V, vx, vy, dt, x1, x2, ObstacleSpace);

  % Add dead reckoning position to Tree and plot it
  if ~isempty(NewBranch)

    %Determine cost of new branch
    Dist2Goal = sqrt((Finish(1)-NewBranch(1))^2+(Finish(2) -
NewBranch(2))^2);
    HDOP = HDOP_Calc(NewBranch,Sub,Sensors);
    flight_time = 0;          %Drifting phase, no flight
```

```
    Cost = ( K1*(HDOP/MaxHDOP) + K2*(Dist2Goal/MaxDistance) +
K3*((flight_power*(flight_time/60))/FlightEnergyMax) -
K4*(solar_energy/24/3600/SolarEnergyMax) )*dt;

    % Add new branch to the tree
    tlength = size(Tree,1);           %Size of tree before element
addition
    Tree(tlength+1,1) = tlength+1;       %Assigns number as newest node
    Tree(tlength+1,2) = NewBranch(1);      %New Node X-position in tree
    Tree(tlength+1,3) = NewBranch(2);      %New Node Y-position in tree
    Tree(tlength+1,4) = Tree(tlength,1);    %Keeps track of parent
    Tree(tlength+1,5) = Cost;          %Cost of branch
    Tree(tlength+1,6) = Tree(tlength,6) + Cost; %Cumulative Cost of
Family (Path)
    Tree(tlength+1,7) = 0;           %Indicator variable for Hopping
behavior added

    % Plot new branch
    figure(1);
    plothandle(length(Tree)) = plot([Tree(tlength,2) NewBranch(1)],
[Tree(tlength,3) NewBranch(2)],'k');

    % If branch got close enough to the Goal, terminate script
    if (sqrt((NewBranch(1)-Finish(1))^2+(NewBranch(2)-Finish(2))^2) <
distThresh )
      display('Success!')
      Success = 1; %Indicator variable
      break
    end

    % Evaluate for negative gradient in distance to the Goal
    if exist('LastDist2Goal','var')
      RangeGradient = (LastDist2Goal - Dist2Goal)/dt;
    end
    LastDist2Goal = sqrt((Finish(1)-NewBranch(1))^2+(Finish(2) -
NewBranch(2))^2);

  end

  % Iterate counter for Dist2Goal gradient <= .1 or unsuccessful branch
  if i ~=1 && (RangeGradient <= .1 || isempty(NewBranch));
    Counter = Counter + 1;
  else
    Counter = 0;
  end

  % If Dist2Go gradient is < .1 for greater than 10 iterations, Hop.
  if Counter >= 10;
    Hop = 1;      % Hop indicator for plotting
    Counter = 0;    % Resets counter

    % Pseudo-randomly select Target point in grid space
    RandomNumber = rand;
    if RandomNumber < GoalProbability
```

```
      Target = Finish;
    elseif RandomNumber > GoalProbability
      Target = random('Uniform',-1,1,1,2) * MapLength;
    end

    % Find the ID of closest nodes
    NodeDistances = sqrt((Tree(:,2)-Target(1)).^2+(Tree(:,3)-
Target(2)).^2);
    [Distance, ClosestNodeNum] = sort(NodeDistances);

    % Determine optimal node of "N" closest points
    try
      for j = 1:N;
        NearNode = Tree(ClosestNodeNum(j),2:3);
        Dist2Goal = sqrt((Finish(1)-NearNode(1))^2+(Finish(2) -
NearNode(2))^2);
        HDOP = HDOP_Calc(NearNode,Sub,Sensors);
        flight_time = MaxFlightTime;  %Nominal minutes of flight
required
        J(j) = ( K1*(HDOP/MaxHDOP) + K2*(Dist2Goal/MaxDistance) +
K3*((flight_power*(flight_time/60))/FlightEnergyMax) -
K4*(solar_energy/24/3600/SolarEnergyMax) )*dt;
      end
    catch  %This statement prevents error when less than "N" branches
in tree
    end

    % Select minimum cost & corresponding optimal node
    [~,I]=min(J);
    OptimalNode = Tree(ClosestNodeNum(I),2:3);
    Dist2Goal = sqrt((Finish(1)-OptimalNode(1))^2+(Finish(2) -
OptimalNode(2))^2);

    % Conduct hop maneuver to Target (or Goal, if close)
    if Dist2Goal <= HopDistance;    %If Goal is in range, fly to
Finish.
      NewBranch = Finish;
      flight_time = MaxFlightTime;  %Nominal minutes of flight required
      Cost =
(Dist2Goal/HopDistance)*K3*((flight_power*(flight_time/60))/FlightEnerg
yMax);
    else
      V = HopDistance/dt;       %Forcing velocity to account for longer
flight time than sim timestep
      NewBranch = DR_RRTstar_Extend(xlimits, ylimits, OptimalNode,
Target, V, vx, vy, dt, x1, x2, ObstacleSpace);
      flight_time = MaxFlightTime;  %Nominal minutes of flight required
      Cost = K3*((flight_power*(flight_time/60))/FlightEnergyMax);
    end

    % Add new branch and Rewire the tree.
    if ~isempty(NewBranch);
      tlength = size(Tree,1);                %Size of tree before
element addition
```

```
      Tree(tlength+1,1) = tlength+1;             %Assigns number as
newest node
      Tree(tlength+1,2) = NewBranch(1);            %New Node X-position
in tree
      Tree(tlength+1,3) = NewBranch(2);            %New Node Y-position
in tree
      Tree(tlength+1,4) = ClosestNodeNum(I);         %Keeps track of
parent
      Tree(tlength+1,5) = Cost;                %Cost of branch
      Tree(tlength+1,6) = Tree(ClosestNodeNum(I),6) + Cost;
%Cumulative Cost of Family (Path)
      Tree(tlength+1,7) = 1;                 %Indicator variable for
Hopping behavior added

      % Plot flight arc.
      figure(1);
      Xa = [OptimalNode(1) mean([OptimalNode(1) NewBranch(1)])
NewBranch(1)];   %[Start Intermediate End]
      Ya = [OptimalNode(2) mean([OptimalNode(2) NewBranch(2)])+1e3
NewBranch(2)]; %[Start Intermediate End]
      t = 1:numel(Xa);
      ts = linspace(min(t),max(t),numel(Xa)*10);
      xx = spline(t,Xa,ts); yy = spline(t,Ya,ts);               %
Create close mesh of points
      plothandle(length(Tree)) = plot(xx,yy,'g');               % Plot
flight curve
      plot([OptimalNode(1) NewBranch(1)],[OptimalNode(2)
NewBranch(2)],'or') % Plot endpoints for reference

      % If branch got close enough to the target, terminate script
      if i ~= 1 && (sqrt((NewBranch(1)-Finish(1))^2+(NewBranch(2)-
Finish(2))^2) < distThresh ||...
          (NewBranch(1) == Finish(1) && NewBranch(2) == Finish(2)))
          display('Success!')
          Success = 1; %Indicator variable
          break
      end

      % Rewire tree (Makes Newest Node the parent of a Near Node if
lower cost)
      try
        % Find the ID of closest nodes to New Branch
        NodeDistances = sqrt((Tree(1:end-1,2)-
NewBranch(1)).^2+(Tree(1:end-1,3)-NewBranch(2)).^2);
        [Distance, ClosestNodeNum] = sort(NodeDistances); %Sort
distance array from min to max

        for j = 1:N;

          % Evaluate cost function which is proportional to max flight
distance.
          flight_time = MaxFlightTime;
```

```
          J(j) =
(Distance(j)/HopDistance)*K3*((flight_power*(flight_time/60))/FlightEne
rgyMax);

          % If cost-effective, within RewireRadius, and already a Hop
point itself (take-off or landing), make Newest Node its parent
          if (Tree(length(Tree),6) + J(j)) < Tree(ClosestNodeNum(j),6)
&& Distance(j) <= RewireRadius && (Tree(ClosestNodeNum(j),7) ~= 0 ||
Tree(Tree(ClosestNodeNum(j),4),7) ~= 0)
            display('Rewire Made!')

            %Reassign parent, update cost
            deltaCost = (Tree(length(Tree),6) + J(j)) -
Tree(ClosestNodeNum(j),6); %Retain cost delta for rewire use
            Tree(ClosestNodeNum(j),4) = Tree(length(Tree),1);
%Replace parent
            Tree(ClosestNodeNum(j),5) = J(j);              %Assign new
cost
            Tree(ClosestNodeNum(j),6) = Tree(length(Tree),6) + J(j);
%Replace cumulative cost
            Tree(ClosestNodeNum(j),7) = 1;               %Indicator
variable for Hopping behavior added

            %Update plot
            delete(plothandle(ClosestNodeNum(j)))            %Deletes
old branch
            Xa = [NewBranch(1) mean([NewBranch(1)
Tree(ClosestNodeNum(j),2)]) Tree(ClosestNodeNum(j),2)];   %[Start
Intermediate End]
            Ya = [NewBranch(2) mean([NewBranch(2)
Tree(ClosestNodeNum(j),3)])+1e3 Tree(ClosestNodeNum(j),3)];  %[Start
Intermediate End]
            t = 1:numel(Xa); ts = linspace(min(t),max(t),numel(Xa)*10);
            xx = spline(t,Xa,ts); yy = spline(t,Ya,ts);         %
Create close mesh of points
            plothandle(length(Tree)) = plot(xx,yy,'g--');        % Plot
flight curve
            plot([NewBranch(1) Tree(ClosestNodeNum(j),2)],[NewBranch(2)
Tree(ClosestNodeNum(j),3)],'or') % Plot endpoints for reference

            %Update cumulative cost for the children
            ChildrenIndex = find(Tree(:,4) == ClosestNodeNum(j));
%Find children of reassigned node
            while ~isempty(ChildrenIndex)
              Tree(ChildrenIndex,6) = Tree(ChildrenIndex, 6) +
deltaCost; %Update children's new cumulative cost
              ChildrenIndex = find(Tree(:,4) == ChildrenIndex);
            end
          end
        end
      catch  %This statement prevents error when less than "N" branches
in tree
      end
```

```
    end
  else
    Hop = 0;
  end
end

toc   %Outputs computation time for simulation

%If unsuccessful, output statement
if (Success ~= 1 && i == iterations)
  fprintf('After %d iterations, goal was not reached...\n',
iterations);
  legend('Current','Start','Finish','Branches')
end
%% Plot Results & Assemble Final Tree

% Plots final path from Finish back to Start
if exist('Success','var'),
  PreviousX = Finish(1);
  PreviousY = Finish(2);
  CurrentNode = Tree(length(Tree),:);

  % Backtrack the successful path
  FlightEnergyFinal = 0;
  for k = 1:iterations;

    % Find parent of current node
    ParentNodeID = CurrentNode(4);

    % Plot branch of best path
    if Tree(CurrentNode(1),7) == 0;  %Neglects flight arcs
      plot([PreviousX Tree(ParentNodeID,2)],...
        [PreviousY Tree(ParentNodeID,3)], 'r', 'LineWidth', 2);
      hold on;
    else
      plot([PreviousX Tree(ParentNodeID,2)],...
        [PreviousY Tree(ParentNodeID,3)], 'r-.', 'LineWidth', 2);
      FlightEnergyFinal = FlightEnergyFinal +
flight_power*(sqrt((Tree(ParentNodeID,2)-
PreviousX)^2+(Tree(ParentNodeID,3) - PreviousY )^2)/2000/20);
    end

    % Change current node to its parent
    CurrentNode = Tree(ParentNodeID,:);
    PreviousX = CurrentNode(2);
    PreviousY = CurrentNode(3);

    % If startpoint was reached, terminate plot script
    if ( (PreviousX == Start(1)) && (PreviousY == Start(2)) )
      break;
    end
  end

  % Display preliminary information
```

```
  fprintf('\nNumber of iterations to find solution: %d \n', i);
  fprintf('Cumulative "Cost" of Path: %.3f \n', Tree(end,6));
  fprintf('Energy Used in Path for Flight: %.3f Watt-hours \n',...
    FlightEnergyFinal);
  fprintf('Energy Used in Direct Flight: %.3f Watt-hours \n',...
    flight_power*(sqrt((Finish(1)-Start(1))^2+...
    (Finish(2) - Start(2))^2)/2000/20)); %Distance converted to nm and
divided by nominal 20kts speed
  fprintf('Potential Energy Gain from Solar: %.3f Watt-hours \n',...
    k*dt*solar_energy/24/3600);
  legend('Current','Start','Finish','Branches')

  % Assembles final tree for aggregation step where combine multiple
hops
  for k = 1:iterations;
    if k ~= 1;
      FinalTree(k,:) = Tree(ParentNodeID,:);
      if ((Tree(ParentNodeID,2) == Start(1)) && (Tree(ParentNodeID,3)
== Start(2)))
        break;  % If startpoint was reached, terminate tree assembly
loop
      end
    else
      FinalTree(k,:) = Tree(length(Tree),:);
    end
    ParentNodeID = FinalTree(k,4);
  end
  fprintf('Estimated time to traverse path: %.3f hours\n',
(length(FinalTree)-sum(FinalTree(:,7)))*dt/3600 +
sum(FinalTree(:,7))*MaxFlightTime/60);

  % Aggregates consecutive hops
  k=1;
  for j = 1:length(FinalTree);

    k = k + 1; %Separate counter used due to necessary resizing of
FinalTree
    if k > length(FinalTree)  %Ends loop once we reach the end of the
resized FinalTree
      break;
    end
    if FinalTree(k,7) ~= 0 && FinalTree(k-1,7) ~= 0 %If "hop-indicator"
variable is nonzero, indicating flight

      %Reassign parent (Note: Does not recompute individual or
cumulative cost)
      FinalTree(k-1,4) = FinalTree(k+1,1);

      %Remove redundant row from Final Tree
      FinalTree = [FinalTree(1:k-1,:); FinalTree(k+1:end,:)];
      k=k-1; %Roll back counter to account for removal of redundant row
    end
  end
```

```
  % Plot the revised path
  fprintf('\n Aggregating consecutive hop steps... \n')
  pause(2)
  FlightEnergyFinalRevised = 0;
  for k = 2:length(FinalTree);

    if FinalTree(k-1,7) == 0;  %Plot drift paths
      plot([FinalTree(k,2) FinalTree(k-1,2)],[FinalTree(k,3)
FinalTree(k-1,3)], 'm', 'LineWidth', 2);
      hold on;
    else  %Plot flight arcs
      plot([FinalTree(k-1,2) FinalTree(k,2)],[FinalTree(k-1,3)
FinalTree(k,3)], 'm-.', 'LineWidth', 2);
      Xa = [FinalTree(k-1,2) mean([FinalTree(k-1,2) FinalTree(k,2)])
FinalTree(k,2)];    %[Start Intermediate End]
      Ya = [FinalTree(k-1,3) mean([FinalTree(k-1,3)
FinalTree(k,3)])+1e3 FinalTree(k,3)];   %[Start Intermediate End]
      t = 1:numel(Xa);
      ts = linspace(min(t),max(t),numel(Xa)*10);
      xx = spline(t,Xa,ts); yy = spline(t,Ya,ts);
% Create close mesh of points
      plot(xx,yy,'m');                                    % Plot flight
curve
      plot([FinalTree(k-1,2) FinalTree(k,2)],[FinalTree(k-1,3)
FinalTree(k,3)],'or')     % Plot endpoints for reference
      FlightEnergyFinalRevised = FlightEnergyFinalRevised +
flight_power*(sqrt((FinalTree(k,2)-FinalTree(k-1,2))^2+(FinalTree(k,3)-
FinalTree(k-1,3))^2)/2000/20);
    end
  end
  fprintf('Energy Used in Path for Flight after Aggregation: %.3f Watt-
hours \n', FlightEnergyFinalRevised);
  fprintf('Energy Used in Direct Flight: %.3f Watt-hours \n',...
    flight_power*(sqrt((Finish(1)-Start(1))^2+...
    (Finish(2) - Start(2))^2)/2000/20)); %Distance converted to nm and
divided by nominal 20kts speed
end
```

```
function [HDOP] = HDOP_Calc(QuadPosition,Sub,Sensors)

%========INFO========

%LT Chase Dillard
%Dead-Reckoning Rapidly-Exploring Random Tree Star (DR-RRT*) Algorithm
%HDOP Function
%11/26/2014

%Implemented in MATLAB. Version: MATLAB R2012b
%Based upon:  Chapter 8 of J. A. Farrell, Aided Navigation: GPS with
%        High Rate Sensors. New York: McGraw Hill, 2008.

%This function: Calculates the Horizontal Dilution of Precision
%using the assumed postion of the target and the other sensors in the
%group.

%====================



% Import positions of sensors
Quad1=QuadPosition; %Position of the node being analyzed
Quad2=Sensors(1:2); %Position of other players, initialized at start of
DR-RRT*
Quad3=Sensors(3:4);
Quad4=Sensors(5:6);

% Jacobian of nonlinear measurement for bearing-only tracking
R1=(Sub(2)-Quad1(2))^2+(Sub(1)-Quad1(1))^2;
R2=(Sub(2)-Quad2(2))^2+(Sub(1)-Quad2(1))^2;
R3=(Sub(2)-Quad3(2))^2+(Sub(1)-Quad3(1))^2;
R4=(Sub(2)-Quad4(2))^2+(Sub(1)-Quad4(1))^2;

H = [-(Sub(2)-Quad1(2))/R1, (Sub(1)-Quad1(1))/R1;
   -(Sub(2)-Quad2(2))/R2, (Sub(1)-Quad2(1))/R2;
   -(Sub(2)-Quad3(2))/R3, (Sub(1)-Quad3(1))/R3;
   -(Sub(2)-Quad4(2))/R4, (Sub(1)-Quad4(1))/R4];

%H matrix for Range-Only HDOP calculation, based upon GPS pseudorange
model
% H_HDOP = [(x_hat(2)-sensor(2))/sqrt(R1), (x_hat(1)-
sensor(1))/sqrt(R1);
%      (x_hat(2)-sensor(4))/sqrt(R2), (x_hat(1)-sensor(3))/sqrt(R2);
%      (x_hat(2)-sensor(6))/sqrt(R3), (x_hat(1)-sensor(5))/sqrt(R3);
%      (x_hat(2)-sensor(8))/sqrt(R4), (x_hat(1)-sensor(7))/sqrt(R4)];

HDOP = sqrt(trace(inv(H'*H)));

end
```

```
function [Cell, V, vx, vy] = CellFinder(TestNode, x1, x2, x1d, x2d)

%========INFO========

%LT Chase Dillard
%Dead-Reckoning Rapidly-Exploring Random Tree Star (DR-RRT*) Algorithm
%Cell Finder Function
%11/26/2014

%Implemented in MATLAB. Version: MATLAB R2012b

%This function: Determines cell placement based upon the (x,y) position
%of a selected test node. Exports the component ocean current velocity
%and speed at that location

%====================

% Determines the grid placement of current node
  for j = 1:2; %Variable switches between grids x1 and x2
    for k = 1:20
      if j == 1 && TestNode(j) >= x1(1,k) && TestNode(j) < x1(1,k+1)
        Cell(2) = k; %Column of x1
        break
      end

      if j == 2 && TestNode(j) >= x2(k,1) && TestNode(j) < x2(k+1,1)
        Cell(1) = k; %Row of x2
        break
      end
    end
  end

  for j = 1:2; %Capture endpoints not accounted for above
    if Cell(j) == 0
      Cell(j) = length(x1);
    end
  end
% Determines the components & magnitude of the ocean current at current
node
vx = x1d(Cell(1),Cell(2));
vy = x2d(Cell(1),Cell(2));
V = sqrt((vx)^2 + (vy)^2);
end
```

```
function NewBranch = DR_RRTstar_Extend(xlimits, ylimits, OptimalNode,
Target, V, vx, vy, dt, x1, x2, ObstacleSpace)

%========INFO========

%LT Chase Dillard
%Dead-Reckoning Rapidly-Exploring Random Tree Star (DR-RRT*) Algorithm
%Extend Function
%11/26/2014

%Implemented in MATLAB. Version: MATLAB R2012b

%This function: Extends a branch in the direction of the Target. Checks
to
%see if the endpoint of this branch falls within an obstacle or outside
%the limits of the map. If it does, export NewBranch is empty. If not,
%export NewBranch is an (x,y) position.

%====================

if isnumeric(Target);  %Flight extension

  % Use basic trig to determine heading
  Opposite = abs(OptimalNode(1) - Target(1));
  Adjacent = abs(OptimalNode(2) - Target(2));
  Hypotenuse = sqrt((OptimalNode(1)-Target(1))^2+(OptimalNode(2)-
Target(2))^2);
  sinPsi = Opposite/Hypotenuse;
  cosPsi = Adjacent/Hypotenuse;

  % Determine position delta
  dx = V*sinPsi*dt;
  dy = V*cosPsi*dt;

  % Conduct Euler integration of point-mass kinematics
  if ( ((OptimalNode(1) - Target(1)) < 0) && ((OptimalNode(2) -
Target(2)) < 0) )
    NewBranch(1) = (OptimalNode(1) + dx);
    NewBranch(2) = (OptimalNode(2) + dy);

  elseif ( ((OptimalNode(1) - Target(1)) > 0) && ((OptimalNode(2) -
Target(2)) < 0) )
    NewBranch(1) = (OptimalNode(1) - dx);
    NewBranch(2) = (OptimalNode(2) + dy);

  elseif ( ((OptimalNode(1) - Target(1)) < 0) && ((OptimalNode(2) -
Target(2)) > 0) )
    NewBranch(1) = (OptimalNode(1) + dx);
    NewBranch(2) = (OptimalNode(2) - dy);

  else
    NewBranch(1) = (OptimalNode(1) - dx);
    NewBranch(2) = (OptimalNode(2) - dy);
```

```
  end

else          %Dead-Reckoning extension
  dx = vx*dt;
  dy = vy*dt;
  NewBranch(1) = (OptimalNode(1) + dx);
  NewBranch(2) = (OptimalNode(2) + dy);
end

% Check if it's not out of bounds
if ( (NewBranch(1) > xlimits(2)) || (NewBranch(1) < xlimits(1)) ||
(NewBranch(2) > ylimits(2))) || (NewBranch(2) < ylimits(1))
  NewBranch = [];
end

% Determines the grid placement of current node
if ~isempty(NewBranch)
  for j = 1:2; %Variable switches between grids x1 and x2
    for k = 1:20
      if j == 1 && NewBranch(j) >= x1(1,k) && NewBranch(j) < x1(1,k+1)
        Cell(2) = k; %Column of x1
        break
      end

      if j == 2 && NewBranch(j) >= x2(k,1) && NewBranch(j) < x2(k+1,1)
        Cell(1) = length(x2)+1-k; %Row of x2.
        break
      end
    end
  end

  %Capture endpoints not accounted for above
  for j = 1:2; %Capture endpoints not accounted for above
    if Cell(j) == 0
      Cell(j) = length(x1);
    end
  end

  % Check if that coordinate is not covered by an obstacle
  if (ObstacleSpace(Cell(1),Cell(2)) == 1)
    NewBranch = [];
  end
end
end
```

LIST OF REFERENCES

[1] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB.* Berlin, Germany: Springer, 2011, pp. 3–532.

[2] S. Pappas. (2011, Nov. 15). Newly discovered ‘alien’ sea worms ride the current. [Online]. Available: http://www.livescience.com/17054-sea-worms-drift-ocean.html

[3] P. Rincon. (2013, Jun. 18). ‘Hoff’ yeti crab hitched ride on ocean super-highway. [Online]. Available: http://www.bbc.com/news/science-environment-22952728

[4] H. Medwin and C. S. Clay, *Fundamentals of Acoustical Oceanography*. San Diego, CA: Academic Press, 1998, pp. 1–102.

[5] T. Qu, “Role of ocean dynamics in determining the mean seasonal cycle of the South China Sea surface temperatures,” *Journal of Geophysical Research*, vol. 106, 2001, pp. 6943–6956.

[6] R. W. Prouty, *Helicopter Performance, Stability, and Control*, 1st ed. Boston, MA: PWS Eng, 1986, ch. 1, pp. 1-10.

[7] Spreading Wings S800—Specs. (n.d.) [Online]. Available: http://www.dji.com/product/spreading-wings-s800/spec.

[8] M. Clifford, C. Horton and J. Schmitz, “SWAFS: Shallow water analysis and forecast system,” *OCEANS ‘94 Oceans Engineering for Today’s Technology and Tomorrow’s Preservation*. Brest, France, 1994.

[9] F. L. Bub. (2011). The Status of Ocean Modeling at the Naval Oceanographic Office (NAVOCEANO) [Powerpoint]. [Online]. Available: https://hycom.org/attachments/101_F.Bub.pdf.

[10] I. Gloza, K. Buszman and R. Jozwiak, “Tracking underwater noise sources with the use of a passive method,” *Acta Physica Polonica A*, vol. 123, no. 6, 2013, pp. 1090–1093.

[11] E. Dahlberg, A. Lauberts, R. K. Lennartsson, M. J. Levonen and L. Persson, “Underwater target tracking by means of acoustic and electromagnetic data fusion,” in *Proceedings of the 9th International Conference on Information Fusion*, Florence, Italy, 2006, pp. 1–7.

[12] K. L. Cockrell and H. Schmidt, “Robust passive range estimation using the waveguide invariant,” *J. Acoust. Soc. Am.*, vol. 127, pp. 2780–2789, Jan., 2010.

[13] S. Chun and K. Kim, “Passive acoustic source tracking using underwater distributed sensors,” *Intl. J. of Dist. Sensor Networks*, vol. 2013, Oct. 2013.

[14] T. Dutoit and F. Marques, *Applied Signal Processing: A MATLAB-Based Proof of Concept.* New York, NY: Springer, 2009, pp. 198–199.

[15] Acousonde. (n.d.) [Online]. Available: http://www.acousonde.com/downloads/Acousonde3A_Brochure.pdf

[16] D. Simon, *Optimal State Estimation: Kalman, H-Infinity and Nonlinear Approaches*. Hoboken, NJ: John Wiley & Sons, 2006.

[17] G. Welch and G. Bishop, “An introduction to the Kalman filter,” presented at *SIGGRAPH 2001*, Los Angeles, CA, 2001.

[18] J. A. Farrell, *Aided Navigation: GPS with High Rate Sensors.* New York, NY: McGraw Hill, 2008.

[19] S. J. Julier and J. K. Uhlmann, “Unscented filtering and nonlinear estimation,” in *Proc. of the IEEE*, 2004, vol. 92, no. 3, pp. 401–422.

[20] E. A. Wan and R. van der Merwe, “The unscented Kalman filter for nonlinear estimation,” Oregon Grad. Inst. of Sci. and Tech., Beaverton, OR, 2000.

[21] S. Karaman and E. Frazzoli, “Optimal kinodynamic motion planning using incremental sampling-based methods,” presented at the *49th IEEE Conf. on Decision and Cntrl.*, Atlanta, GA, 2010.

[22] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” Iowa State University, Ames, IA, 1998.

[23] C. Yang, E. Blasch and I. Kadar, “Geometric factors in target positioning and tracking,” presented at the *12th Intl. Conf. on Inf. Fusion*, Seattle, WA, 2009.

[24] J. D. Bard and F. M. Ham, “Time difference of arrival dilution of precision and applications,” *IEEE Transactions on Signal Processing*, vol. 47, Feb., 1999.

[25] V. Tas, “Optimal Use of TDO Geo-Location Techniques Within the Mountainous Terrain of Turkey,” M.S. thesis, Dept. Info. Sci., Naval Postgraduate School, Monterey, CA, 2012.

[26] J. Nasir, F. Islam and Y. Ayaz, “Adaptive rapidly-exploring-random-tree-star (RRT*)-smart: algorithm characteristics and behavior analysis in complex environments,” *Asia-Pacific J. of Inf. Tech. and Multimedia*, vol. 2, p. 39, Dec, 2013.

[27] D. J. Webb and J. van den Berg. (2012). Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints. [Online]. Available: http://arxiv.org/pdf/1205.5088.pdf.

[28] J. W. Loeve, “Finding time-optimal trajectories for the resonating arm using the RRT* algorithm,” M.S. thesis, Dept. Mech., Maritime and Mat. Eng., Delft Univ. of Tech., Delft, Netherlands, 2012.

[29] A. Lavin, “A pareto optimal D* search algorithm for multiobjective path planning,” to be presented at *IEEE Intl. Conf. on Robotics and Automation*, Seattle, WA, 2015.

[30] P. Tebbutt, J. Wood and M. King. (2002). *The Vicon manual*. Vicon Motion Systems, Lake Forest, CA. [Online]. Available: http://www.biomech.uottawa.ca/english/teaching/apa6905/lectures/vicon_manual_v1_2.pdf

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California